APPENDIX A

IMPLICIT MAPPING OF TECHNOLOGY INDEPENDENT NETWORK TO LIBRARY CELLS

THIERRY BESSON

M-7928 US

```
/*-----------------------------------------------------------------------*/
/*                                                                       */
/*     File:           gnlutil.c                                         */
/*     Version:        1.1                                               */
/*     Modifications:  -                                                 */
/*     Documentation:  -                                                 */
/*                                                                       */
/*     Description:                                                      */
/*                                                                       */
/*-----------------------------------------------------------------------*/


#include <stdio.h>
#include <sys/time.h>

#ifdef MEMORY           /* If one wants memory statistics for SUN */
#include <malloc.h>
#endif

#include "blist.h"
#include "gnl.h"
#include "gnlmint.h"
#include "bbdd.h"
#include "libc_mem.h"
#include "libc_api.h"
#include "gnllibc.h"
#include "gnloption.h"

#include "blist.e"


/*-----------------------------------------------------------------------*/
/* EXTERN                                                                */
/*-----------------------------------------------------------------------*/
extern GNL_ENV    G_GnlEnv;
extern LIBC_PIN GnlGetPinCellWithName ();

/*-----------------------------------------------------------------------*/
/* GnlFreeVar                                                            */
/*-----------------------------------------------------------------------*/
/* Physical destruction of the Gnl Variable 'Var'.                       */
/*-----------------------------------------------------------------------*/
void GnlFreeVar (Var)
   GNL_VAR  Var;
{

   free (GnlVarName (Var));
   if (GnlVarLocation (Var))
      free (GnlVarLocation (Var));
   free (Var);

}

/*-----------------------------------------------------------------------*/
/* SetGnlVarLineNumber                                                   */
/*-----------------------------------------------------------------------*/
/* This procedure builds the string representing the line number where  */
```

```
/* the var 'Var' is defined and stores it in the field 'GnlVarLocation' */
/* of the Var. This string can generally represent several line numbers */
/* of different source files.                                           */
/*----------------------------------------------------------------------*/
GNL_STATUS SetGnlVarLineNumber (Var, Line)
   GNL_VAR  Var;
   int            Line;
{
   char           *CompLocString;
   char           *LocString;


   if ((LocString = (char*)calloc (128, sizeof(char))) == NULL)
      return (GNL_MEMORY_FULL);

   sprintf (LocString, "%d ", Line);

   if ((CompLocString = (char*)
                    calloc (strlen(LocString)+1, sizeof(char))) == NULL)
      return (GNL_MEMORY_FULL);

   sprintf (CompLocString, "%s", LocString);

   free (LocString);

   SetGnlVarLocation (Var, CompLocString);

   return (GNL_OK);
}

/*----------------------------------------------------------------------*/
/* GnlVarLineNumber                                                     */
/*----------------------------------------------------------------------*/
/* This procedure extracts the line number of the var 'Var' from the    */
/* location string stored in field 'GnlVarLocation'. By calling this    */
/* procedure we assume that the string stores only one integer (should  */
/* have been set by 'SetGnlVarLineNumber' above).                       */
/*----------------------------------------------------------------------*/
int GnlVarLineNumber (Var)
   GNL_VAR      Var;
{
   char         LocString[128];
   int            i;
   int            Line;


   if (!GnlVarLocation (Var))
      return (0);

   i = 0;
   while ((i<strlen (GnlVarLocation (Var)) &&
          (GnlVarLocation (Var)[i] != ' ')))
         {
          LocString[i] = GnlVarLocation (Var)[i];
          i++;
         }
```

```
   LocString[i] = '\0';

   Line = atoi (LocString);

   return (Line);
}


/*-------------------------------------------------------------------------*/
/* GnlResetVarFunction                                                     */
/*-------------------------------------------------------------------------*/
/* Reset the field 'GnlVarFunction' of the variable 'Var'.                 */
/*-------------------------------------------------------------------------*/
void GnlResetVarFunction (Var)
   GNL_VAR  Var;
{
   SetGnlVarFunction (Var, NULL);
}


/*-------------------------------------------------------------------------*/
/* GnlFunctionFree                                                         */
/*-------------------------------------------------------------------------*/
/* Procedure which physically free the structure pointed by Function       */
/* which is of type GNL_FUNCTION_REC                                       */
/*-------------------------------------------------------------------------*/
void GnlFunctionFree (Function)
   GNL_FUNCTION   Function;
{
   free ((char*) Function);
}

/* ----------------------------------------------------------------------- */
/* GnlSystemGetTimeOfDay                                                   */
/* ----------------------------------------------------------------------- */
void GnlSystemGetTimeOfDay (Seconds, MicroSeconds)
  long *Seconds;                /* seconds since Jan. 1, 1970            */
  long *MicroSeconds;           /* and microseconds                */

{
  struct timeval tp;
  struct timezone tzp;

  (void) gettimeofday (&tp, &tzp);
  *Seconds = tp.tv_sec;
  *MicroSeconds = tp.tv_usec;
}

/* ----------------------------------------------------------------------- */
/* GnlTimeGetAbsoluteTime                                                    */
/* ----------------------------------------------------------------------- */
/* returns the elapsed time in seconds since  January, 1, 1970             */
/* ----------------------------------------------------------------------- */
long GnlTimeGetAbsoluteTime ()
{
  long      Seconds;
  long      MicroSeconds;
```

```
  GnlSystemGetTimeOfDay (&Seconds, &MicroSeconds);
  return (Seconds);
}

/* ------------------------------------------------------------------ */
/* GnlSystemGetLocalTime                                              */
/* ------------------------------------------------------------------ */
/* Seconds (0 - 59)                                                   */
/* Minutes (0 - 59)                                                   */
/* Hours (0 - 23)                                                     */
/* Day : day of month (1 - 31)                                        */
/* Month : month of year (0 - 11)                                     */
/* Year : year - 1900                                                 */
/* ------------------------------------------------------------------ */
void GnlSystemGetLocalTime (Year, Month, Day, Hours, Minutes, Seconds)
   int *Year;
   int *Month;
   int *Day;
   int *Hours;
   int *Minutes;
   int *Seconds;
{
   struct tm    *CurDate;
   long         Time;


   Time = GnlTimeGetAbsoluteTime ();

   CurDate = localtime (&Time);
   *Year = CurDate->tm_year;
   *Month = CurDate->tm_mon + 1;
   *Day = CurDate->tm_mday;
   *Hours = CurDate->tm_hour;
   *Minutes = CurDate->tm_min;
   *Seconds = CurDate->tm_sec;
}

/* ------------------------------------------------------------------ */
/* GnlPrintDate                                                       */
/* ------------------------------------------------------------------ */
/* Prints the date and time under format "04/20/99 @ 11:15:30" in the */
/* file 'File'.                                                       */
/* ------------------------------------------------------------------ */
void GnlPrintDate (File)
   FILE          *File;
{
   int  Year;
   int  Month;
   int  Day;
   int  Hours;
   int  Minutes;
   int  Seconds;


   GnlSystemGetLocalTime (&Year, &Month, &Day, &Hours, &Minutes, &Seconds);
```

```
   fprintf (File, "%.2d/%.2d/%d @ %.2d:%.2d:%.2d", Month, Day, Year,
            Hours, Minutes, Seconds);

}

/*--------------------------------------------------------------------------*/
/* GnlVarIsIO                                                    */
/*--------------------------------------------------------------------------*/
/* Returns 1 if the current var 'Var' is either a primary INPUT, or a   */
/* primary OUTPUT or a primary INOUT                                 */
/*--------------------------------------------------------------------------*/
int GnlVarIsIO (Var)
   GNL_VAR      Var;
{
   if (GnlVarDir (Var) == GNL_VAR_INPUT)
      return (1);
   if (GnlVarDir (Var) == GNL_VAR_OUTPUT)
      return (1);
   if (GnlVarDir (Var) == GNL_VAR_INOUT)
      return (1);

   return (0);
}

/*--------------------------------------------------------------------------*/
/* GnlVarIsPrimary                                               */
/*--------------------------------------------------------------------------*/
/* Returns 1 if the current var 'Var' is a primary var that means which */
/* cannot be removed at anytime (except for GNL_VAR_LOCAL_WIRING when   */
/* they are re-injected during a hierarchycal flattening).          */
/*--------------------------------------------------------------------------*/
int GnlVarIsPrimary (Var)
   GNL_VAR  Var;
{
   if (GnlVarDir (Var) == GNL_VAR_INPUT)
      return (1);
   if (GnlVarDir (Var) == GNL_VAR_OUTPUT)
      return (1);
   if (GnlVarDir (Var) == GNL_VAR_INOUT)
      return (1);
   if (GnlVarDir (Var) == GNL_VAR_LOCAL_WIRING)
      return (1);

   return (0);
}


/*--------------------------------------------------------------------------*/
/* GnlVarRangeSize                                               */
/*--------------------------------------------------------------------------*/
/* Function returning the size of the range of the var 'Var'. A single  */
/* bit var has a size range of 1 and for instance a bus (3 downto 0)    */
/* has a range size of 4.                                        */
/*--------------------------------------------------------------------------*/
int GnlVarRangeSize (Var)
   GNL_VAR  Var;
{
```

```
   if (GnlVarMsb (Var) >= GnlVarLsb (Var))
      return (GnlVarMsb (Var) - GnlVarLsb (Var) + 1);
   else
      return (GnlVarLsb (Var) - GnlVarMsb (Var) + 1);
}


/*-------------------------------------------------------------------------*/
/* GnlDirName                                                              */
/*-------------------------------------------------------------------------*/
/* Returns a verilog-semantic string corresponding to the direction of  */
/* the var 'Var'.                                                          */
/*-------------------------------------------------------------------------*/
char *GnlDirName (Var)
   GNL_VAR  Var;
{
  switch (GnlVarDir (Var)) {
      case GNL_VAR_INPUT: return ("input");
      case GNL_VAR_OUTPUT: return ("output");
      case GNL_VAR_INOUT: return ("inout");

      default:
             fprintf (stderr,
                 " ERROR: cannot find the direction of variable '%s'\n",
                 GnlVarName (Var));
             exit (1);
      }
}

/*-------------------------------------------------------------------------*/
/* GnlSeqComponentIsDFF                                                    */
/*-------------------------------------------------------------------------*/
/* Returns 1 if the seqnetial component 'SeqComponent' is a Flip-Flop   */
/* and 0 otherwise.                                                        */
/*-------------------------------------------------------------------------*/
int GnlSeqComponentIsDFF (SeqComponent)
   GNL_SEQUENTIAL_COMPONENT   SeqComponent;
{

   if ((GnlSequentialCompoOp (SeqComponent) == GNL_DFF) ||
       (GnlSequentialCompoOp (SeqComponent) == GNL_DFFX) ||
       (GnlSequentialCompoOp (SeqComponent) == GNL_DFF0) ||
       (GnlSequentialCompoOp (SeqComponent) == GNL_DFF1))
      return (1);

   return (0);
}

/*-------------------------------------------------------------------------*/
/* GnlSeqComponentIsLATCH                                                  */
/*-------------------------------------------------------------------------*/
/* Returns 1 if the seqnetial component 'SeqComponent' is a LATCH */
/* and 0 otherwise.                                                        */
/*-------------------------------------------------------------------------*/
int GnlSeqComponentIsLATCH (SeqComponent)
   GNL_SEQUENTIAL_COMPONENT   SeqComponent;
{
```

```
   if ((GnlSequentialCompoOp (SeqComponent) == GNL_LATCH) ||
       (GnlSequentialCompoOp (SeqComponent) == GNL_LATCHX) ||
       (GnlSequentialCompoOp (SeqComponent) == GNL_LATCH1) ||
       (GnlSequentialCompoOp (SeqComponent) == GNL_LATCH0))
      return (1);

   return (0);
}

/*-------------------------------------------------------------------------*/
/* GnlVarRangeUndefined                                                    */
/*-------------------------------------------------------------------------*/
/* returns 1 if the range of the var 'Var' is not defined (e.g. its Msb */
/* and Lsb are equal to -1).                                               */
/*-------------------------------------------------------------------------*/
int GnlVarRangeUndefined (Var)
   GNL_VAR  Var;
{
  if ((GnlVarMsb (Var) == -1) && (GnlVarLsb (Var) == -1))
     return (1);

  return (0);
}

/*-------------------------------------------------------------------------*/
/* GnlVarGetStrFromRange                                                   */
/*-------------------------------------------------------------------------*/
/* Returns a string thru 'RangeStr' which is the verilog-format   */
/* representation of the range of the variable 'Var'.             */
/*-------------------------------------------------------------------------*/
GNL_STATUS GnlVarGetStrFromRange (Var, RangeStr)
   GNL_VAR  Var;
   char           **RangeStr;
{
   if (GnlVarRangeUndefined (Var))
      {
       *RangeStr = NULL;
       return (GNL_OK);
      }

   if ((*RangeStr = (char*)GNL_CALLOC (256, sizeof(char))) == NULL)
      return (GNL_MEMORY_FULL);

   sprintf ((*RangeStr), "[%d:%d]", GnlVarMsb (Var), GnlVarLsb (Var));

   return (GNL_OK);
}


/*-------------------------------------------------------------------------*/
/* GnlNameIndexName                                                        */
/*-------------------------------------------------------------------------*/
/* This function creates physically a new name which is the concatanate */
/* of '[', the Index and ']'.                                              */
/* Example: Name = "bus" , Index = 0 then *NewStr = "bus[0]".          */
/*-------------------------------------------------------------------------*/
```

```
GNL_STATUS GnlNameIndexName (Str, Index, NewStr)
   char           *Str;
   int            Index;
   char           **NewStr;
{
   char     *Str1;
   char     *Str2;


   if (GnlEnvInputFormat() == GNL_INPUT_VLG)
      {
       if (GnlStrAppendStrCopy (Str, "[", &Str1))
          return (GNL_MEMORY_FULL);
      }
   else
      {
       if (GnlStrAppendStrCopy (Str, "(", &Str1))
          return (GNL_MEMORY_FULL);
      }

   if (GnlStrAppendIntCopy (Str1, Index, &Str2))
      return (GNL_MEMORY_FULL);

   if (GnlEnvInputFormat() == GNL_INPUT_VLG)
      {
       if (GnlStrAppendStrCopy (Str2, "]", NewStr))
          return (GNL_MEMORY_FULL);
      }
   else
      {
       if (GnlStrAppendStrCopy (Str2, ")", NewStr))
          return (GNL_MEMORY_FULL);
      }

   free (Str1);
   free (Str2);

   return (GNL_OK);
}

/*-------------------------------------------------------------------------*/
/* GnlVarIndexName                                                         */
/*-------------------------------------------------------------------------*/
/* This function creates physically a new name which is the concatanate */
/* of '[', the Index and ']' or '(' ')' in fsm mode.                       */
/* Example: Var = "bus" , Index = 0 then *NewStr = "bus[0]".               */
/*-------------------------------------------------------------------------*/
GNL_STATUS GnlVarIndexName (Var, Index, NewStr)
   GNL_VAR  Var;
   int            Index;
   char           **NewStr;
{

   if (GnlNameIndexName (GnlVarName (Var), Index, NewStr))
      return (GNL_MEMORY_FULL);

   return (GNL_OK);
```

```
}

/*-------------------------------------------------------------------------*/
/* GnlVarStrNameStrIndex                                                   */
/*-------------------------------------------------------------------------*/
/* Extracts Base name and Index from the name of an Index Var. The name */
/* of an index Var should of the forme: BASE_NAME[Index]          */
/*-------------------------------------------------------------------------*/
GNL_STATUS GnlVarStrNameStrIndex (Str, VarName, Index)
   char     *Str;
   char     **VarName;
   int      *Index;
{
   int      LengthStr;
   char     *IndexName;
   int      i;
   int      j;


   LengthStr = strlen (Str);
   if (Str[LengthStr-1] != ']')           /* Possibly an index Var      */
      {
       if (GnlStrCopy (Str, VarName))
          return (GNL_MEMORY_FULL);
       *Index = -1;
       return (GNL_OK);
      }

  if ((*VarName = (char*)GNL_CALLOC (LengthStr, sizeof(char))) == NULL)
     return (GNL_MEMORY_FULL);

  if ((IndexName = (char*)GNL_CALLOC (LengthStr, sizeof(char))) == NULL)
     return (GNL_MEMORY_FULL);

  i = 0;
  while (i < LengthStr-1)
      {
       if (Str[i] == '[')
           break;
       (*VarName)[i] = Str[i];
       i++;
      }
  (*VarName)[i] = '\0';

  j = 0;
  while (i < LengthStr-1)
      {
       if (Str[i] == ']')
           break;
       IndexName[j] = Str[i];
       j++;
       i++;
      }
  IndexName[j] = '\0';

  *Index = atoi (IndexName);
  free (IndexName);
```

```
  return (GNL_OK);
}


/*-------------------------------------------------------------------------*/
/* GnlStrIndexName                                                         */
/*-------------------------------------------------------------------------*/
/* This returns a string thru 'NewStr' which is the concatenation of the*/
/* signal name 'Str' '[' 'Index' ']'.                                      */
/* Example: Str = "x", Index = 2 ---> NewStr = "x[2]"                      */
/*-------------------------------------------------------------------------*/
GNL_STATUS GnlStrIndexName (Str, Index, NewStr)
   char           *Str;
   int           Index;
   char          **NewStr;
{
   char *Str1;
   char *Str2;


   if (GnlStrAppendStrCopy (Str, "[", &Str1))
      return (GNL_MEMORY_FULL);

   if (GnlStrAppendIntCopy (Str1, Index, &Str2))
      return (GNL_MEMORY_FULL);

   if (GnlStrAppendStrCopy (Str2, "]", NewStr))
      return (GNL_MEMORY_FULL);

   free (Str1);
   free (Str2);

   return (GNL_OK);
}

/*-------------------------------------------------------------------------*/
/* GnlGetVarFromName                                                       */
/*-------------------------------------------------------------------------*/
/* We look for the object GNL_VAR whose name is 'Name' in the Hash Table*/
/* names. GNL_OK is returned if the object is found and it is assecible */
/* thru 'Var'. If does not exist then GNL_VAR_NOT_EXISTS is returned.   */
/*-------------------------------------------------------------------------*/
GNL_STATUS GnlGetVarFromName (Gnl, Name, Var)
   GNL            Gnl;
   char           *Name;
   GNL_VAR  *Var;
{
   BLIST        NewList;
   GNL_VAR      VarI;
   unsigned int Key;
   int          i;
   BLIST        HashTableNames;


   HashTableNames = GnlHashNames (Gnl);
```

```
    /* The list GnlHashNames (Gnl) is of size HASH_TABLE_NAMES_SIZE      */
    Key = KeyOfName (Name, BListSize (HashTableNames));

    if (HashTableNames->Adress[Key] == (int)NULL)
       return (GNL_VAR_NOT_EXISTS);

    NewList = (BLIST)(HashTableNames->Adress[Key]);

    for (i=0; i<BListSize (NewList); i++)
        {
         VarI = (GNL_VAR)BListElt (NewList, i);

         if (!strcmp (GnlVarName (VarI), Name))
            {
             *Var = VarI;
             return (GNL_OK);
            }
        }

     return (GNL_VAR_NOT_EXISTS);
}

/*-------------------------------------------------------------------------*/
/* GnlResetVarHook                                                         */
/*-------------------------------------------------------------------------*/
/* Reset the field 'GnlVarHook' of all the Var of 'Gnl'.                   */
/*-------------------------------------------------------------------------*/
void GnlResetVarHook (Gnl)
   GNL  Gnl;
{
   int          i;
   int          j;
   BLIST        BucketI;
   GNL_VAR      VarJ;
   BLIST        HashTableNames;


   HashTableNames = GnlHashNames (Gnl);
   for (i=0; i<BListSize (HashTableNames); i++)
       {
        BucketI = (BLIST)BListElt (HashTableNames, i);
        for (j=0; j<BListSize (BucketI); j++)
            {
             VarJ = (GNL_VAR)BListElt (BucketI, j);
             SetGnlVarHook (VarJ, NULL);
            }
       }
}

/*-------------------------------------------------------------------------*/
/* GnlResetVarTag                                                          */
/*-------------------------------------------------------------------------*/
/* Reset the field 'GnlVarTag' of all the Var of 'Gnl'.                    */
/*-------------------------------------------------------------------------*/
void GnlResetVarTag (Gnl)
   GNL  Gnl;
{
```

```
    int          i;
    int          j;
    BLIST        BucketI;
    GNL_VAR      VarJ;
    BLIST        HashTableNames;


    HashTableNames = GnlHashNames (Gnl);
    for (i=0; i<BListSize (HashTableNames); i++)
        {
         BucketI = (BLIST)BListElt (HashTableNames, i);
         for (j=0; j<BListSize (BucketI); j++)
             {
              VarJ = (GNL_VAR)BListElt (BucketI, j);
              SetGnlVarTag (VarJ, 0);
             }
        }
}

/*-------------------------------------------------------------------------*/
/* GnlResetVarDads                                                         */
/*-------------------------------------------------------------------------*/
/* Reset the field 'GnlVarDads' of all the Var of 'Gnl'.                   */
/*-------------------------------------------------------------------------*/
void GnlResetVarDads (Gnl)
    GNL  Gnl;
{
    int          i;
    int          j;
    BLIST        BucketI;
    GNL_VAR      VarJ;
    BLIST        HashTableNames;


    HashTableNames = GnlHashNames (Gnl);
    for (i=0; i<BListSize (HashTableNames); i++)
        {
         BucketI = (BLIST)BListElt (HashTableNames, i);
         for (j=0; j<BListSize (BucketI); j++)
             {
              VarJ = (GNL_VAR)BListElt (BucketI, j);
              SetGnlVarDads (VarJ, NULL);
             }
        }
}

/*-------------------------------------------------------------------------*/
/* GnlResetGnlNetworkTagInGnl                                              */
/*-------------------------------------------------------------------------*/
void GnlResetGnlNetworkTagInGnl (Gnl)
    GNL             Gnl;
{
    BLIST           Components;
    int                   i;
    GNL_COMPONENT   ComponentI;
    GNL_USER_COMPONENT    UserCompoI;
    GNL                   GnlCompoI;
```

```
    SetGnlTag (Gnl, 0);

    Components = GnlComponents (Gnl);

    for (i=0; i<BListSize (Components); i++)
        {
         ComponentI = (GNL_COMPONENT)BListElt (Components, i);
         if (GnlComponentType (ComponentI) != GNL_USER_COMPO)
            continue;

         UserCompoI = (GNL_USER_COMPONENT)ComponentI;
         GnlCompoI = GnlUserComponentGnlDef (UserCompoI);

         if (!GnlCompoI)
            continue;

         GnlResetGnlNetworkTagInGnl (GnlCompoI);
        }
}

/*-------------------------------------------------------------------------*/
/* GnlResetGnlNetworkTag                                                   */
/*-------------------------------------------------------------------------*/
/* Resets the field tag of each Gnl of the current network 'Nw'.  */
/*-------------------------------------------------------------------------*/
void GnlResetGnlNetworkTag (Nw)
   GNL_NETWORK    Nw;
{
   GNL      TopGnl;

   TopGnl = GnlNetworkTopGnl (Nw);
   GnlResetGnlNetworkTagInGnl (TopGnl);
}


/*-------------------------------------------------------------------------*/
/* GnlAddVarInHashTablenames                                               */
/*-------------------------------------------------------------------------*/
/* We add the GNL_VAR 'Var' in the HashTableNames of 'Gnl' according to */
/* the Variable Name. It returns:                                          */
/*          - GNL_MEMORY_FULL if no more memory to extend the     */
/*            Hash Table Names.                                            */
/*          - GNL_VAR_EXISTS if the 'Var' is already present in   */
/*            the Hash Table Names.                                        */
/*          - GNL_OK if the 'Var' was not present and has been    */
/*            added.                                                */
/*-------------------------------------------------------------------------*/
GNL_STATUS GnlAddVarInHashTablenames (Gnl, Var)
   GNL            Gnl;
   GNL_VAR  Var;
{
   BLIST        NewList;
   GNL_VAR      VarI;
   unsigned int Key;
   int          i;
```

```
   BLIST        HashTableNames;
   GNL_STATUS   GnlStatus;


   HashTableNames = GnlHashNames (Gnl);

   /* The list GnlHashNames (Gnl) is of size HASH_TABLE_NAMES_SIZE      */
   Key = KeyOfName (GnlVarName (Var), BListSize (HashTableNames));

   if (HashTableNames->Adress[Key] == (int)NULL)
      {
       if (BListCreateWithSize (1, &NewList))
          return (GNL_MEMORY_FULL);

       HashTableNames->Adress[Key] = (int)NewList;
      }

   NewList = (BLIST)(HashTableNames->Adress[Key]);

   for (i=0; i<BListSize (NewList); i++)
       {
        VarI = (GNL_VAR)BListElt (NewList, i);
        if (!strcmp (GnlVarName (VarI), GnlVarName (Var)))
            return (GNL_VAR_EXISTS);
       }

   if (BListAddElt (NewList, (int)Var))
      return (GNL_MEMORY_FULL);

   return (GNL_OK);
}

/*-------------------------------------------------------------------------*/
/* GnlAddCompoNameInHashTablenames                                         */
/*-------------------------------------------------------------------------*/
/* We add the string 'CompoName' in the HashTableCompoNames of 'Gnl'       */
/* if it does not exist. If not '*CompoName' takes the value of the        */
/* already existing compo name.                                            */
/*-------------------------------------------------------------------------*/
GNL_STATUS GnlAddCompoNameInHashTablenames (Gnl, CompoName)
   GNL            Gnl;
   char           **CompoName;
{
   BLIST        NewList;
   char           *CompoNameI;
   unsigned int Key;
   int          i;
   BLIST        HashTableCompoNames;
   GNL_STATUS   GnlStatus;


   HashTableCompoNames = GnlHashCompoNames (Gnl);

   /* The list GnlHashCompoNames (Gnl) is of size                          */
   /* HASH_TABLE_COMPO_NAMES_SIZE.                                         */
```

```
    Key = KeyOfName ((*CompoName), HASH_TABLE_COMPO_NAMES_SIZE);

    if (HashTableCompoNames->Adress[Key] == (int)NULL)
       {
        if (BListCreateWithSize (1, &NewList))
           return (GNL_MEMORY_FULL);

        HashTableCompoNames->Adress[Key] = (int)NewList;
       }

    NewList = (BLIST)(HashTableCompoNames->Adress[Key]);

    for (i=0; i<BListSize (NewList); i++)
        {
         CompoNameI = (char*)BListElt (NewList, i);
         if (!strcmp (CompoNameI, (*CompoName)))
            {
             free ((*CompoName));
            *CompoName = CompoNameI;
             return (GNL_OK);
            }
        }

    if (BListAddElt (NewList, (int)(*CompoName)))
       return (GNL_MEMORY_FULL);


    return (GNL_OK);
}


/*-------------------------------------------------------------------------*/
/* GnlCreateUniqueVarWithNoTest                                            */
/*-------------------------------------------------------------------------*/
/* This procedure creates a GNL_VAR object and stores it in the Hash       */
/* table names of 'Gnl'. The Var name is based on the name 'BaseName'      */
/* but may be not unique since the unicity test is by-passed. This         */
/* function must be invoked if the caller knows that the base name he      */
/* is giving is unique by construction.                                    */
/*-------------------------------------------------------------------------*/
static char G_NewUniqueName[528];
GNL_STATUS GnlCreateUniqueVarWithNoTest (Gnl, BaseName, NewVar)
   GNL            Gnl;
   char           *BaseName;
   GNL_VAR  *NewVar;
{
   GNL_STATUS     GnlStatus;
   char           *NewName;


   SetGnlUniqueId (Gnl, GnlUniqueId (Gnl)+1);
   sprintf (G_NewUniqueName, "\\%s_%d", BaseName, GnlUniqueId (Gnl));

   if ((GnlStatus = GnlVarCreateAndAddInHashTableWithNoTest (Gnl,
                           G_NewUniqueName, NewVar)))
      return (GNL_MEMORY_FULL);
```

```
   return (GNL_OK);
}

/*-------------------------------------------------------------------------*/
/* GnlCreateUniqueVarWithNoTestGnlId                                       */
/*-------------------------------------------------------------------------*/
/* This procedure creates a GNL_VAR object and stores it in the Hash       */
/* table names of 'Gnl'. The Var name is based on the name 'BaseName'      */
/* but may be not unique since the unicity test is by-passed. This         */
/* function must be invoked if the caller knows that the base name he      */
/* is giving is unique by construction.                                    */
/* The difference with 'GnlCreateUniqueVarWithNoTest' is that the Gnl      */
/* Id used is the one given by 'GnlId' and not the one from 'Gnl'.         */
/*-------------------------------------------------------------------------*/
GNL_STATUS GnlCreateUniqueVarWithNoTestGnlId (GnlId, Gnl, BaseName,
                                              NewVar)
   GNL           GnlId;
   GNL           Gnl;
   char          *BaseName;
   GNL_VAR       *NewVar;
{
   GNL_STATUS    GnlStatus;
   char          *NewName;


   SetGnlUniqueId (GnlId, GnlUniqueId (GnlId)+1);
   sprintf (G_NewUniqueName, "\\%s_%d", BaseName, GnlUniqueId (GnlId));

   if ((GnlStatus = GnlVarCreateAndAddInHashTableWithNoTest (Gnl,
                                 G_NewUniqueName, NewVar)))
      return (GNL_MEMORY_FULL);

   return (GNL_OK);
}


/*-------------------------------------------------------------------------*/
/* GnlCreateUniqueVar                                                      */
/*-------------------------------------------------------------------------*/
/* This procedure creates a Unique GNL_VAR object and stores it in the     */
/* Hash table names of 'Gnl'. The Var name is based on the name            */
/* 'BaseName' and is unique. It can be either 'BaseName' if this one is    */
/* unique in 'Gn' or a derivated name. A derivated name is the             */
/* 'BaseName' concatanate with a unique Id.                                */
/*-------------------------------------------------------------------------*/
GNL_STATUS GnlCreateUniqueVar (Gnl, BaseName, NewVar)
   GNL           Gnl;
   char          *BaseName;
   GNL_VAR  *NewVar;
{
   int           ConflictName;
   GNL_STATUS    GnlStatus;


   /* If no based name specified then we take the Gnl one                  */
   if (!BaseName)
      BaseName = GnlName (Gnl);
```

```
   if ((GnlStatus = GnlVarCreateAndAddInHashTable (Gnl, BaseName,
                                                   NewVar)))
      {
       if (GnlStatus == GNL_MEMORY_FULL)
          return (GNL_MEMORY_FULL);
       ConflictName = 1;
      }
   else
      return (GNL_OK);

   while (ConflictName)
       {
      ConflictName = 0;
      SetGnlUniqueId (Gnl, GnlUniqueId (Gnl)+1);

      sprintf (G_NewUniqueName, "\\%s_%d", BaseName, GnlUniqueId (Gnl));

      if ((GnlStatus = GnlVarCreateAndAddInHashTable (Gnl,
                               G_NewUniqueName, NewVar)))
           {
            if (GnlStatus == GNL_MEMORY_FULL)
               return (GNL_MEMORY_FULL);
          ConflictName = 1;
         }
       }

   return (GNL_OK);
}

/*-------------------------------------------------------------------------*/
/* GnlVarNameExistsInGnl                                                   */
/*-------------------------------------------------------------------------*/
/* This function returns 1 if the name 'VarName' is already defined        */
/* among the name in the gnl 'Gnl'.                                        */
/*-------------------------------------------------------------------------*/
int GnlVarNameExistsInGnl (Gnl, VarName)
   GNL             Gnl;
   char            *VarName;
{
   BLIST        NewList;
   GNL_VAR      VarI;
   unsigned int Key;
   int          i;
   BLIST        HashTableNames;
   GNL_STATUS   GnlStatus;



   HashTableNames = GnlHashNames (Gnl);

   /* The list GnlHashNames (Gnl) is of size HASH_TABLE_NAMES_SIZE      */
   Key = KeyOfName (VarName, BListSize (HashTableNames));

   if (HashTableNames->Adress[Key] == (int)NULL)
      {
       return (0);
```

```
        }

    NewList = (BLIST)(HashTableNames->Adress[Key]);

    for (i=0; i<BListSize (NewList); i++)
        {
         VarI = (GNL_VAR)BListElt (NewList, i);
         if (!strcmp (GnlVarName (VarI), VarName))
             return (1);
        }

    return (0);
}


/*-------------------------------------------------------------------------*/
/* GnlCreateUniqueVarName                                                  */
/*-------------------------------------------------------------------------*/
GNL_STATUS GnlCreateUniqueVarName (Gnl, BaseName, NewName)
    GNL          Gnl;
    char         *BaseName;
    char         **NewName;
{

    if (!GnlVarNameExistsInGnl (Gnl, BaseName))
       {
        *NewName = BaseName;
        return (GNL_OK);
       }

    while (1)
        {
         SetGnlUniqueId (Gnl, GnlUniqueId (Gnl)+1);

         sprintf (G_NewUniqueName, "%s_%d", BaseName, GnlUniqueId (Gnl));

         if (!GnlVarNameExistsInGnl (Gnl, G_NewUniqueName))
            {
             free (BaseName);
             if (GnlStrCopy (G_NewUniqueName, NewName))
                return (GNL_MEMORY_FULL);
             return (GNL_OK);
            }

        }

    return (GNL_OK);
}

/*-------------------------------------------------------------------------*/
/* GnlRemoveVarFromGnlLocals                                               */
/*-------------------------------------------------------------------------*/
/* Removes the variable 'Var' from the list of locals of 'Gnl'.            */
/*-------------------------------------------------------------------------*/
void GnlRemoveVarFromGnlLocals (Gnl, Var)
    GNL          Gnl;
    GNL_VAR      Var;
```

```
{
   int          i;
   GNL_VAR      VarI;


   for (i=0; i<BListSize (GnlLocals (Gnl)); i++)
       {
        VarI = (GNL_VAR)BListElt (GnlLocals (Gnl), i);
        if (Var == VarI)
           {
            BListDelInsert (GnlLocals (Gnl), i+1);
            break;
           }
       }

}


/*-------------------------------------------------------------------------*/
/* GnlRemoveVarFromGnlHashTableName                                        */
/*-------------------------------------------------------------------------*/
/* This procedure removes the var 'Var' from the hash table names of       */
/* 'Gnl'.                                                                  */
/*-------------------------------------------------------------------------*/
void GnlRemoveVarFromGnlHashTableName (Gnl, Var)
   GNL            Gnl;
   GNL_VAR  Var;
{
   BLIST    HashTableNames;
   BLIST    Bucket;
   int            i;
   GNL_VAR  VarI;
   unsigned int Key;


   HashTableNames = GnlHashNames (Gnl);

   Key = KeyOfName (GnlVarName (Var), BListSize (HashTableNames));

   /* This var does not appear since no bucket.                            */
   if (HashTableNames->Adress[Key] == (int)NULL)
      return;

   Bucket = (BLIST)HashTableNames->Adress[Key];

   for (i=0; i<BListSize (Bucket); i++)
       {
      VarI = (GNL_VAR)BListElt (Bucket, i);
      if (VarI == Var)
           {
            BListDelInsert (Bucket, i+1);
         return;
        }
       }
}

/*-------------------------------------------------------------------------*/
```

```
/* GnlNameFromOp                                                    */
/*------------------------------------------------------------------*/
char *GnlNameFromOp (Op)
   GNL_OP   Op;
{

   switch (Op) {
      case GNL_AND: return (".");
      case GNL_OR: return ("+");
      case GNL_NOT: return ("!");
      case GNL_XOR: return ("^");
      case GNL_NAND: return ("*");
      case GNL_NOR: return ("#");
      case GNL_XNOR: return ("@");

      default: return ("??");
      }
}


/*------------------------------------------------------------------*/
/* GnlVarIsVss                                                      */
/*------------------------------------------------------------------*/
/* returns 1 if the var is a the predefined var VSS (0).            */
/*------------------------------------------------------------------*/
int GnlVarIsVss (Var)
   GNL_VAR  Var;
{

   if (GnlVarName (Var)[0] == '0')
      return (1);

   return (0);
}


/*------------------------------------------------------------------*/
/* GnlVarIsVdd                                                      */
/*------------------------------------------------------------------*/
/* returns 1 if the var is a the predefined var VDD (1).            */
/*------------------------------------------------------------------*/
int GnlVarIsVdd (Var)
   GNL_VAR  Var;
{

   if (GnlVarName (Var)[0] == '1')
      return (1);

   return (0);
}

/*------------------------------------------------------------------*/
/* GnlNodeIsVss                                                     */
/*------------------------------------------------------------------*/
/* A node is a Vdd in two cases:                                    */
/*     - either it is a GNL_CONSTANTE operator with GnlNodeSons = 0,    */
/*     - or a GNL_VARIABLE operator and the variable is a GnlVarIsVss   */
```

```
/*-------------------------------------------------------------------------*/
int GnlNodeIsVss (Node)
   GNL_NODE     Node;
{

   if (GnlNodeOp (Node) == GNL_VARIABLE)
      {
       return (GnlVarIsVss ((GNL_VAR)GnlNodeSons (Node)));
      }

   if (GnlNodeOp (Node) == GNL_CONSTANTE)
      {
       return (GnlNodeSons (Node) == (BLIST)0);
      }

   return (0);
}

/*-------------------------------------------------------------------------*/
/* GnlNodeIsVdd                                                            */
/*-------------------------------------------------------------------------*/
/* A node is a Vdd in two cases:                                           */
/*      - either it is a GNL_CONSTANTE operator with GnlNodeSons = 1,      */
/*      - or a GNL_VARIABLE operator and the variable is a GnlVarIsVdd     */
/*-------------------------------------------------------------------------*/
int GnlNodeIsVdd (Node)
   GNL_NODE     Node;
{

   if (GnlNodeOp (Node) == GNL_VARIABLE)
      {
       return (GnlVarIsVdd ((GNL_VAR)GnlNodeSons (Node)));
      }

   if (GnlNodeOp (Node) == GNL_CONSTANTE)
      {
       return (GnlNodeSons (Node) == (BLIST)1);
      }

   return (0);

}

/*-------------------------------------------------------------------------*/
/* GnlVarIsSignal                                                          */
/*-------------------------------------------------------------------------*/
int GnlVarIsSignal (Var)
   GNL_VAR  Var;
{

   if (GnlVarIsVdd (Var) || GnlVarIsVss (Var))
      return (0);

   return (1);
}

/*-------------------------------------------------------------------------*/
```

```
/* GnlNodeCopy                                                              */
/*--------------------------------------------------------------------------*/
/* Copy the Node 'Node' at only one level, i.e. the structure and */
/* eventually the list of sons. The sons elements are not physically    */
/* duplicated and are simply shared.                                        */
/*--------------------------------------------------------------------------*/
GNL_STATUS GnlNodeCopy (Gnl, Node, NodeCopy)
   GNL            Gnl;
   GNL_NODE Node;
   GNL_NODE *NodeCopy;
{
   int            i;
   BLIST    NewSons;


   if (GnlCreateNode (Gnl, GnlNodeOp (Node), NodeCopy))
      return (GNL_MEMORY_FULL);
   SetGnlNodeLineNumber (*NodeCopy, GnlNodeLineNumber (Node));

   if ((GnlNodeOp (*NodeCopy) == GNL_VARIABLE) ||
       (GnlNodeOp (*NodeCopy) == GNL_CONSTANTE) ||
       (GnlNodeOp (*NodeCopy) == GNL_WIRE))
      {
       /* Only the Value of 'GnlNodeSons (Node)' is relevant.          */
       SetGnlNodeSons (*NodeCopy, GnlNodeSons (Node));
       return (GNL_OK);
      }

   /* Otherwise we need to duplicate the list of sons.                  */
   /* The list is duplicated but the pointed elements are physically the*/
   /* same.                                                  */
   if (BListCreateWithSize (BListSize (GnlNodeSons (Node)), &NewSons))
      return (GNL_MEMORY_FULL);

   for (i=0; i<BListSize (GnlNodeSons (Node)); i++)
       {
        if (BListAddElt (NewSons, BListElt (GnlNodeSons (Node), i)))
           return (GNL_MEMORY_FULL);
       }

   SetGnlNodeSons (*NodeCopy, NewSons);

   return (GNL_OK);
}

/*--------------------------------------------------------------------------*/
/* GnlNodeCopyRec                                                           */
/*--------------------------------------------------------------------------*/
/* Copy the Node 'Node' and its sub-tree recursively.             */
/*--------------------------------------------------------------------------*/
GNL_STATUS GnlNodeCopyRec (Gnl, Node, NodeCopy)
   GNL          Gnl;
   GNL_NODE     Node;
   GNL_NODE     *NodeCopy;
{
   int            i;
   BLIST    Sons;
```

```
    BLIST    NewList;
    GNL_NODE SonI;
    GNL_NODE NodeCopyI;


    switch (GnlNodeOp (Node)) {
       case GNL_CONSTANTE:
         case GNL_VARIABLE:
         case GNL_WIRE:
              return (GnlNodeCopy (Gnl, Node, NodeCopy));

         default:
              Sons = GnlNodeSons (Node);
              if (BListCreateWithSize (BListSize (Sons), &NewList))
                 return (GNL_MEMORY_FULL);
              if (GnlCreateNode (Gnl, GnlNodeOp (Node), NodeCopy))
                 return (GNL_MEMORY_FULL);
              SetGnlNodeLineNumber (*NodeCopy, GnlNodeLineNumber (Node));
              SetGnlNodeSons (*NodeCopy, NewList);

              for (i=0; i<BListSize (Sons); i++)
                  {
                   SonI = (GNL_NODE)BListElt (Sons, i);
                   if (GnlNodeCopyRec (Gnl, SonI, &NodeCopyI))
                      return (GNL_MEMORY_FULL);
                   if (BListAddElt (NewList, (int)NodeCopyI))
                      return (GNL_MEMORY_FULL);
                  }
              return (GNL_OK);
       }
}


/*-------------------------------------------------------------------------*/
/* PrintNodeRec                                                            */
/*-------------------------------------------------------------------------*/
void PrintNodeRec (Node, Bind)
   GNL_NODE Node;
   int            Bind;
{
   GNL_NODE Son;
   GNL_VAR  Var;
   int            i;


   switch (GnlNodeOp (Node)) {
      case GNL_VARIABLE:
             Var = (GNL_VAR)GnlNodeSons (Node);
             if (GnlVarIsVss (Var))
                fprintf (stderr, "{0}");
             else if (GnlVarIsVdd (Var))
                fprintf (stderr, "{1}");
             else
            {
               if (Bind && GnlVarBindVar (Var))
                Var = GnlVarBindVar (Var);
                 fprintf (stderr, "%s", GnlVarName (Var));
```

```
            }
             return;

      case GNL_NOT:
           Son = (GNL_NODE)BListElt (GnlNodeSons (Node), 0);
             fprintf (stderr, "!(");
             PrintNodeRec (Son, Bind);
             fprintf (stderr, ")");
             return;

      case GNL_AND:
      case GNL_OR:
      case GNL_NOR:
      case GNL_NAND:
      case GNL_XOR:
      case GNL_XNOR:
             fprintf (stderr, "(");
             for (i=0; i<BListSize (GnlNodeSons (Node))-1; i++)
                 {
                  Son = (GNL_NODE)BListElt (GnlNodeSons (Node), i);
                  PrintNodeRec (Son, Bind);
                  fprintf (stderr, "%s", GnlNameFromOp (GnlNodeOp (Node)));
                 }
             Son = (GNL_NODE)BListElt (GnlNodeSons (Node), i);
             PrintNodeRec (Son, Bind);
             fprintf (stderr, ")");
             return;

      case GNL_CONSTANTE:
           if (GnlNodeSons (Node) == (BLIST)1)
            fprintf (stderr, "{1}");
           else
            fprintf (stderr, "{0}");
           return;

      case GNL_WIRE:
           Son = (GNL_NODE)GnlNodeSons (Node);
           PrintNodeRec (Son, Bind);
           return;

      default:
             GnlError (9 /* Unknown node */);
             return;
       }

}

/*-------------------------------------------------------------------------*/
/* GnlPrintNodeRec                                                         */
/*-------------------------------------------------------------------------*/
void GnlPrintNodeRec (File, Node, Bind)
   FILE           *File;
   GNL_NODE Node;
   int            Bind;
{
   GNL_NODE Son;
   GNL_VAR  Var;
```

```
   int            i;


   switch (GnlNodeOp (Node)) {
      case GNL_VARIABLE:
             Var = (GNL_VAR)GnlNodeSons (Node);
             if (GnlVarIsVss (Var))
                fprintf (File, "{0}");
             else if (GnlVarIsVdd (Var))
                fprintf (File, "{1}");
             else
            {
               if (Bind && GnlVarBindVar (Var))
                Var = GnlVarBindVar (Var);
                 fprintf (File, "%s", GnlVarName (Var));
            }
             return;

      case GNL_NOT:
           Son = (GNL_NODE)BListElt (GnlNodeSons (Node), 0);
             fprintf (File, "!(");
             GnlPrintNodeRec (File, Son, Bind);
             fprintf (File, ")");
             return;

      case GNL_AND:
      case GNL_OR:
      case GNL_NOR:
      case GNL_NAND:
      case GNL_XOR:
      case GNL_XNOR:
             fprintf (File, "(");
             for (i=0; i<BListSize (GnlNodeSons (Node))-1; i++)
                 {
                  Son = (GNL_NODE)BListElt (GnlNodeSons (Node), i);
                  GnlPrintNodeRec (File, Son, Bind);
                  fprintf (File, "%s", GnlNameFromOp (GnlNodeOp (Node)));
                 }
             Son = (GNL_NODE)BListElt (GnlNodeSons (Node), i);
             GnlPrintNodeRec (File, Son, Bind);
             fprintf (File, ")");
             return;

      case GNL_CONSTANTE:
           if (GnlNodeSons (Node) == (BLIST)1)
            fprintf (File, "{1}");
           else
            fprintf (File, "{0}");
           return;

      case GNL_WIRE:
           Son = (GNL_NODE)GnlNodeSons (Node);
           GnlPrintNodeRec (File, Son, Bind);
           return;

      default:
             GnlError (9 /* Unknown node */);
```

```
            return;
        }

}


/*-----------------------------------------------------------------------*/
/* GnlPrintNode                                                          */
/*-----------------------------------------------------------------------*/
void GnlPrintNode (File, Node)
   FILE           *File;
   GNL_NODE Node;
{
   GnlPrintNodeRec (File, Node, 0);
}


/*-----------------------------------------------------------------------*/
/* GnlEqualNode                                                          */
/*-----------------------------------------------------------------------*/
int GnlEqualNode (Node1, Node2)
   GNL_NODE Node1;
   GNL_NODE Node2;
{
   int                  i;
   GNL_NODE       Son1;
   GNL_NODE       Son2;


   if (GnlNodeOp (Node1) != GnlNodeOp (Node2))
      return (0);

   if (GnlNodeOp (Node1) == GNL_VARIABLE)
       return (GnlNodeSons (Node1) == GnlNodeSons (Node2));

   if (GnlNodeOp (Node1) == GNL_CONSTANTE)
       return (GnlNodeSons (Node1) == GnlNodeSons (Node2));

   if (BListSize (GnlNodeSons (Node1)) !=
       BListSize (GnlNodeSons (Node2)))
      return (0);

   for (i=0; i<BListSize (GnlNodeSons (Node1)); i++)
       {
        Son1 = (GNL_NODE)BListElt (GnlNodeSons (Node1), i);
        Son2 = (GNL_NODE)BListElt (GnlNodeSons (Node2), i);
        if (!GnlEqualNode (Son1, Son2))
           return (0);
       }

   return (1);
}

/*-----------------------------------------------------------------------*/
/* GnlSeqName                                                            */
/*-----------------------------------------------------------------------*/
char *GnlSeqName (SeqName)
```

```
   GNL_SEQUENTIAL_OP    SeqName;
{

   switch (SeqName) {
      case GNL_DFF: return ("DFF");
      case GNL_DFFX: return ("DFFX");
      case GNL_DFF0: return ("DFF0");
      case GNL_DFF1: return ("DFF1");
      case GNL_LATCH: return ("LATCH");
      case GNL_LATCHX: return ("LATCHX");
      case GNL_LATCH0: return ("LATCH0");
      case GNL_LATCH1: return ("LATCH1");
        default: return ("DFF ??");
       }
}

/*-------------------------------------------------------------------------*/
/* GnlPrintTriStateBox                                                     */
/*-------------------------------------------------------------------------*/
void GnlPrintTriStateBox (File, PBox)
   FILE                        *File;
   GNL_TRISTATE_COMPONENT      PBox;
{

   fprintf (File,
            "TRISTATE %s (q=%s, d=%s, d_pol=%d, en=%s, en_pol=%d)\n",
            GnlTriStateInstName (PBox),
            GnlVarName (GnlTriStateOutput (PBox)),
            GnlVarName (GnlTriStateInput (PBox)),
            GnlTriStateInputPol (PBox),
            GnlVarName (GnlTriStateSelect (PBox)),
            GnlTriStateSelectPol (PBox));
}


/*-------------------------------------------------------------------------*/
/* GnlPrintPredefinedBox                                                   */
/*-------------------------------------------------------------------------*/
void GnlPrintPredefinedBox (File, PBox)
   FILE                        *File;
   GNL_SEQUENTIAL_COMPONENT    PBox;
{
   char           *SetName;
   char           *ResetName;

   if (!GnlSequentialCompoSet (PBox))
      SetName = "";
   else
      SetName = GnlVarName (GnlSequentialCompoSet (PBox));
   if (!GnlSequentialCompoReset (PBox))
      ResetName = "";
   else
      ResetName = GnlVarName (GnlSequentialCompoReset (PBox));

   if (GnlSeqComponentIsDFF (PBox))
      {
       fprintf (File,
```

```
"[%s] %s (q=%s, d=%s, clk=%s, clk_pol=%d, s=%s, s_pol=%d, r=%s, r_pol=%d)\n",
                GnlSequentialCompoInstName (PBox),
                GnlSeqName (GnlSequentialCompoOp(PBox)),
                GnlVarName (GnlSequentialCompoOutput (PBox)),
                GnlVarName (GnlSequentialCompoInput (PBox)),
                GnlVarName (GnlSequentialCompoClock (PBox)),
                GnlSequentialCompoClockPol (PBox),
                SetName,
                GnlSequentialCompoClockPol (PBox),
                ResetName,
                GnlSequentialCompoResetPol (PBox));
      }
   else
      {
       fprintf (File,
"[%s] %s (q=%s, d=%s, clk=%s, clk_pol=%d, s=%s, s_pol=%d, r=%s, r_pol=%d)\n",
                GnlSequentialCompoInstName (PBox),
                GnlSeqName (GnlSequentialCompoOp(PBox)),
                GnlVarName (GnlSequentialCompoOutput (PBox)),
                GnlVarName (GnlSequentialCompoInput (PBox)),
                GnlVarName (GnlSequentialCompoClock (PBox)),
                GnlSequentialCompoClockPol (PBox),
                SetName,
                GnlSequentialCompoClockPol (PBox),
                ResetName,
                GnlSequentialCompoResetPol (PBox));
      }
}

/*-------------------------------------------------------------------------*/
/* GnlPrintUserBox                                                         */
/*-------------------------------------------------------------------------*/
void GnlPrintUserBox (File, UserCompo)
   FILE                 *File;
   GNL_USER_COMPONENT   UserCompo;
{
   int            i;
   GNL_ASSOC      AssocI;
   char           *FormalName;
   char           *ActualName;


   fprintf (File, "[%s] %s (",
            GnlUserComponentName (UserCompo),
            GnlUserComponentInstName (UserCompo));

   for (i=0; i<BListSize (GnlUserComponentInterface (UserCompo))-1; i++)
       {
        AssocI = (GNL_ASSOC)
               BListElt (GnlUserComponentInterface (UserCompo), i);
        if (GnlUserComponentFormalType (UserCompo) == GNL_FORMAL_CHAR)
           FormalName = GnlAssocFormalPort (AssocI);
        else
           FormalName = GnlVarName ((GNL_VAR)GnlAssocFormalPort (AssocI));
        ActualName = GnlVarName (GnlAssocActualPort (AssocI));

        fprintf (File, ".%s(%s), ", FormalName, ActualName);
```

```
        }

    AssocI = (GNL_ASSOC)
              BListElt (GnlUserComponentInterface (UserCompo), i);
    if (GnlUserComponentFormalType (UserCompo) == GNL_FORMAL_CHAR)
       FormalName = GnlAssocFormalPort (AssocI);
    else
       FormalName = GnlVarName ((GNL_VAR)GnlAssocFormalPort (AssocI));
    ActualName = GnlVarName (GnlAssocActualPort (AssocI));

    fprintf (File, ".%s(%s))\n", FormalName, ActualName);

}

/*-------------------------------------------------------------------------*/
/* GnlPrintGnl                                                             */
/*-------------------------------------------------------------------------*/
void GnlPrintGnl (File, Gnl)
    FILE     *File;
    GNL      Gnl;
{
    int                 i;
    GNL_VAR         VarI;
    GNL_VAR         VarJ;
    BLIST           BucketI;
    int                 j;
    BLIST           HashTableNames;
    GNL_FUNCTION        FunctionI;
    BLIST           Components;
    GNL_COMPONENT   ComponentI;


    fprintf (File, "-------------------------------------------------------
\n");
    fprintf (File, "Name = %s\n\n", GnlName (Gnl));
    fprintf (File, "Inputs = ");
    for (i=0; i<BListSize (GnlInputs (Gnl)); i++)
        {
         VarI = (GNL_VAR)BListElt (GnlInputs (Gnl), i);
         fprintf (File, "%s ", GnlVarName (VarI));
        }
    fprintf (File, ";\n");

    fprintf (File, "Outputs = ");
    for (i=0; i<BListSize (GnlOutputs (Gnl)); i++)
        {
         VarI = (GNL_VAR)BListElt (GnlOutputs (Gnl), i);
         fprintf (File, "%s ", GnlVarName (VarI));
        }
    fprintf (File, ";\n");

    fprintf (File, "Locals = ");
    /* Removing the unused GNL_VAR_LOCAL variables                          */
/*
    HashTableNames = GnlHashNames (Gnl);
    for (i=0; i<BListSize (HashTableNames); i++)
        {
```

```
        BucketI = (BLIST)BListElt (HashTableNames, i);
        for (j=0; j<BListSize (BucketI); j++)
            {
             VarJ = (GNL_VAR)BListElt (BucketI, j);
             if ((GnlVarDir (VarJ) == GNL_VAR_LOCAL) ||
                 (GnlVarDir (VarJ) == GNL_VAR_LOCAL_WIRING))
                {
                 fprintf (File, "%s ", GnlVarName (VarJ));
                }
            }
      }
*/
   for (i=0; i<BListSize (GnlLocals (Gnl)); i++)
       {
        VarJ = (GNL_VAR)BListElt (GnlLocals (Gnl), i);
        if (!VarJ)
           fprintf (File, "NULL ");
        else
           fprintf (File, "%s ", GnlVarName (VarJ));
       }
   fprintf (File, ";\n\n");

   /* Printing the Boolean functions ...                    */
   for (i=0; i<BListSize (GnlFunctions (Gnl)); i++)
       {
        VarI = (GNL_VAR)BListElt (GnlFunctions (Gnl), i);
        FunctionI = GnlVarFunction (VarI);

        fprintf (File, "%s = ", GnlVarName (VarI));
        GnlPrintNode (File, GnlFunctionOnSet (FunctionI));
        fprintf (File, ";\n");

        if (GnlFunctionDCSet (FunctionI))
           {
            fprintf (File, "DCSET %s = ", GnlVarName (VarI));
            GnlPrintNode (File, GnlFunctionDCSet (FunctionI));
            fprintf (File, ";\n");
           }
       }


   return;

   Components = GnlComponents (Gnl);
   if (Components && BListSize (Components))
      {
       fprintf (File, "---------------------\n");
       fprintf (File, "WIRING:\n\n");
      }
   for (i=0; i<BListSize (Components); i++)
       {
        ComponentI = (GNL_COMPONENT)BListElt (Components, i);
        switch (GnlComponentType (ComponentI)) {
            case GNL_SEQUENTIAL_COMPO:
                     GnlPrintPredefinedBox (File,
                                (GNL_SEQUENTIAL_COMPONENT)ComponentI);
                     break;
```

```
            case GNL_TRISTATE_COMPO:
                    GnlPrintTriStateBox (File,
                                (GNL_TRISTATE_COMPONENT)ComponentI);
                    break;

            case GNL_USER_COMPO:
                    GnlPrintUserBox (File,
                                (GNL_USER_COMPONENT)ComponentI);
                    break;
            default:
                    break;
            }
        }

   fprintf (File, "-------------------------------------------------------
\n");
}

/*-------------------------------------------------------------------------*/
/* GnlGetGnlNbLiterals                                                     */
/*-------------------------------------------------------------------------*/
/* Returns the number of Literals used in the GNL 'Gnl'.                   */
/*-------------------------------------------------------------------------*/
int GnlGetGnlNbLiterals (Gnl)
   GNL      Gnl;
{
   int            i;
   int            NbLitt;
   GNL_FUNCTION   FunctionI;
   GNL_VAR  VarI;


   NbLitt = 0;
   for (i=0; i<BListSize (GnlFunctions (Gnl)); i++)
       {
        VarI = (GNL_VAR)BListElt (GnlFunctions (Gnl), i);
        FunctionI = GnlVarFunction (VarI);
        NbLitt += GnlNodeNbLitt (GnlFunctionOnSet (FunctionI));
       }

   return (NbLitt);
}

/*-------------------------------------------------------------------------*/
/* GnlNodeNbEquiGates                                                      */
/*-------------------------------------------------------------------------*/
float GnlNodeNbEquiGates (Node)
   GNL_NODE     Node;
{
   int          i;
   float        NbEquiGates;
   GNL_NODE     SonI;


   if (GnlNodeOp (Node) == GNL_CONSTANTE)
      return (0);
```

```
    if (GnlNodeOp (Node) == GNL_VARIABLE)
       return (0);

    if (GnlNodeOp (Node) == GNL_WIRE)
       {
        SonI = (GNL_NODE)GnlNodeSons (Node);
        return (GnlNodeNbEquiGates (SonI));
       }

    if (GnlNodeOp (Node) == GNL_NOT)
        NbEquiGates = 0.5;
    else
       NbEquiGates = BListSize (GnlNodeSons(Node))-1;

    for (i=0; i<BListSize (GnlNodeSons (Node)); i++)
        {
         SonI = (GNL_NODE)BListElt (GnlNodeSons (Node), i);
         NbEquiGates += GnlNodeNbEquiGates (SonI);
        }

    return (NbEquiGates);
}

/*-------------------------------------------------------------------------*/
/* GnlGetNbEquiGatesInPredefinedBox                                        */
/*-------------------------------------------------------------------------*/
float GnlGetNbEquiGatesInPredefinedBox (PBox)
    GNL_SEQUENTIAL_COMPONENT   PBox;
{

    switch (GnlSequentialCompoOp (PBox)) {
       case GNL_DFF:
       case GNL_DFFX:
       case GNL_DFF1:
       case GNL_DFF0:
            return (2.5);      /* Two master-slave latches + Inv.  */

       case GNL_LATCH:
       case GNL_LATCHX:
       case GNL_LATCH1:
       case GNL_LATCH0:
            return (1);  /* Two inverters              */

       }
}

/*-------------------------------------------------------------------------*/
/* GnlGetGnlNbEquiGates                                                    */
/*-------------------------------------------------------------------------*/
/* Returns the number of Equivalent gates in the GNL 'Gnl'.                */
/*-------------------------------------------------------------------------*/
void GnlGetGnlNbEquiGates (Gnl, CombEquiGates, SeqEquiGates)
   GNL      Gnl;
   float    *CombEquiGates;
   float    *SeqEquiGates;
{
```

```
    int             i;
    float           NbEquiGates;
    GNL_FUNCTION    FunctionI;
    GNL_VAR         VarI;
    GNL_COMPONENT   ComponentI;
    BLIST           Components;


    *CombEquiGates = *SeqEquiGates = 0;

    for (i=0; i<BListSize (GnlFunctions (Gnl)); i++)
      {
       VarI = (GNL_VAR)BListElt (GnlFunctions (Gnl), i);
       FunctionI = GnlVarFunction (VarI);
       *CombEquiGates += GnlNodeNbEquiGates (GnlFunctionOnSet (FunctionI));
      }

    Components = GnlComponents (Gnl);
    if (!Components)
       return ;

    for (i=0; i<BListSize (Components); i++)
        {
         ComponentI = (GNL_COMPONENT)BListElt (Components, i);
         if (GnlComponentType (ComponentI) == GNL_SEQUENTIAL_COMPO)
            {
             *SeqEquiGates += GnlGetNbEquiGatesInPredefinedBox (
              (GNL_SEQUENTIAL_COMPONENT)ComponentI);
            }
        }

}

/*-------------------------------------------------------------------------*/
/* GnlGetNbFlipsFlops                                                      */
/*-------------------------------------------------------------------------*/
/* Returns the number of Flips Flops in the GNL 'Gnl'.                     */
/*-------------------------------------------------------------------------*/
int GnlGetNbFlipsFlops (Gnl)
   GNL  Gnl;
{
   int                          NbFlipsFlops;
   int                          i;
   GNL_SEQUENTIAL_COMPONENT     PBox;
   GNL_COMPONENT          ComponentI;
   BLIST                  Components;


   NbFlipsFlops = 0;

   Components = GnlComponents (Gnl);
   if (!Components)
      return (0);

   for (i=0; i<BListSize (Components); i++)
     {
      ComponentI = (GNL_COMPONENT)BListElt (Components, i);
```

```
      if (GnlComponentType (ComponentI) == GNL_SEQUENTIAL_COMPO)
         {
          PBox = (GNL_SEQUENTIAL_COMPONENT)ComponentI;
          if ((GnlSequentialCompoOp (PBox) == GNL_DFF) ||
              (GnlSequentialCompoOp (PBox) == GNL_DFFX) ||
              (GnlSequentialCompoOp (PBox) == GNL_DFF0) ||
              (GnlSequentialCompoOp (PBox) == GNL_DFF1))
             NbFlipsFlops++;
         }
      }

   return (NbFlipsFlops);
}

/*-------------------------------------------------------------------------*/
/* GnlGetNbLatches                                                         */
/*-------------------------------------------------------------------------*/
/* Returns the number of Latches in the GNL 'Gnl'.                         */
/*-------------------------------------------------------------------------*/
int GnlGetNbLatches (Gnl)
   GNL  Gnl;
{
   int                          NbLatches;
   int                          i;
   GNL_SEQUENTIAL_COMPONENT          PBox;
   GNL_COMPONENT                ComponentI;
   BLIST                   Components;


   NbLatches = 0;

   Components = GnlComponents (Gnl);
   if (!Components)
      return (0);

   for (i=0; i<BListSize (Components); i++)
     {
      ComponentI = (GNL_COMPONENT)BListElt (Components, i);
      if (GnlComponentType (ComponentI) == GNL_SEQUENTIAL_COMPO)
         {
          PBox = (GNL_SEQUENTIAL_COMPONENT)ComponentI;
          if ((GnlSequentialCompoOp (PBox) == GNL_LATCH) ||
              (GnlSequentialCompoOp (PBox) == GNL_LATCHX) ||
              (GnlSequentialCompoOp (PBox) == GNL_LATCH0) ||
              (GnlSequentialCompoOp (PBox) == GNL_LATCH1))
             NbLatches++;
         }
     }

   return (NbLatches);
}

/*-------------------------------------------------------------------------*/
/* GnlGetNbTriSTates                                                       */
/*-------------------------------------------------------------------------*/
/* Returns the number of TriStates in the GNL 'Gnl'.                       */
/*-------------------------------------------------------------------------*/
```

```
int GnlGetNbTriStates (Gnl)
   GNL  Gnl;
{
   int                          NbTriStates;
   int                          i;
   GNL_TRISTATE_COMPONENT       PBox;
   GNL_COMPONENT                ComponentI;
   BLIST                  Components;


   NbTriStates = 0;

   Components = GnlComponents (Gnl);
   if (!Components)
      return (0);

   for (i=0; i<BListSize (Components); i++)
     {
      ComponentI = (GNL_COMPONENT)BListElt (Components, i);
      if (GnlComponentType (ComponentI) == GNL_TRISTATE_COMPO)
          {
           PBox = (GNL_TRISTATE_COMPONENT)ComponentI;
           NbTriStates++;
          }
     }

   return (NbTriStates);
}

/*-------------------------------------------------------------------------*/
/* GnlGetNbBuffers                                                         */
/*-------------------------------------------------------------------------*/
/* Returns the number of Buffers in the GNL 'Gnl'.                         */
/*-------------------------------------------------------------------------*/
int GnlGetNbBuffers (Gnl)
   GNL  Gnl;
{
   int                           NbBuffers;
   int                           i;
   GNL_BUF_COMPONENT            PBox;
   GNL_COMPONENT                 ComponentI;
   BLIST                         Components;


   NbBuffers = 0;

   Components = GnlComponents (Gnl);
   if (!Components)
      return (0);

   for (i=0; i<BListSize (Components); i++)
     {
      ComponentI = (GNL_COMPONENT)BListElt (Components, i);
      if (GnlComponentType (ComponentI) == GNL_BUF_COMPO)
          {
           PBox = (GNL_BUF_COMPONENT)ComponentI;
           NbBuffers++;
```

```
            }
        }

    return (NbBuffers);
}

/*-------------------------------------------------------------------------*/
/* GnlResetNodeHook                                                        */
/*-------------------------------------------------------------------------*/
/* Reset 'Hook' field of 'Node' and recursively on the Sons. If the        */
/* 'Node' has a 0 Hook then we assume it is the same for its sons.         */
/*-------------------------------------------------------------------------*/
void GnlResetNodeHook (Node)
    GNL_NODE Node;
{
    int             i;
    GNL_NODE SonI;


    if (Node == NULL)
        return;

    if ((GnlNodeOp (Node) == GNL_CONSTANTE) ||
        (GnlNodeOp (Node) == GNL_VARIABLE))
        return;

    if (GnlNodeHook (Node) == 0)            /* Already processed previously
        */
        return;

    if (GnlNodeOp (Node) == GNL_WIRE)
        {
        SonI = (GNL_NODE)GnlNodeSons (Node);
        GnlResetNodeHook (SonI);
        return;
        }

    for (i=0; i<BListSize (GnlNodeSons(Node)); i++)
        {
        SonI = (GNL_NODE)BListElt (GnlNodeSons(Node), i);
        GnlResetNodeHook (SonI);
        }

    SetGnlNodeHook (Node, 0);
}

/*-------------------------------------------------------------------------*/
/* GnlResetNodeHookRec                                                     */
/*-------------------------------------------------------------------------*/
/* Reset 'Hook' field of 'Node' and recursively on the Sons. If the        */
/* 'Node' has a 0 Hook then we assume it is the same for its sons.         */
/*-------------------------------------------------------------------------*/
void GnlResetNodeHookRec (Node)
    GNL_NODE Node;
{
    int             i;
    GNL_NODE        SonI;
```

```
    GNL_NODE        NewNode;
    GNL_VAR         Var;
    GNL_FUNCTION          Function;


    if (Node == NULL)
       return;

    if (GnlNodeOp (Node) == GNL_CONSTANTE)
       return;

    if (GnlNodeOp (Node) == GNL_VARIABLE)
       {
        Var = (GNL_VAR)GnlNodeSons (Node);
        Function = GnlVarFunction (Var);
        if (!Function)
           return;

        NewNode = GnlFunctionOnSet (Function);
        GnlResetNodeHook (NewNode);
        return;
       }

    SetGnlNodeHook (Node, NULL);

    for (i=0; i<BListSize (GnlNodeSons(Node)); i++)
        {
         SonI = (GNL_NODE)BListElt (GnlNodeSons(Node), i);
         GnlResetNodeHookRec (SonI);
        }

}


/*-------------------------------------------------------------------------*/
/* GnlResetGnlNodeHook                                                     */
/*-------------------------------------------------------------------------*/
/* This procedure resets all the 'Hook' fields of each GNL_NODE object  */
/*-------------------------------------------------------------------------*/
void GnlResetGnlNodeHook (Gnl)
   GNL      Gnl;
{
   int             i;
   GNL_FUNCTION    FunctionI;
   GNL_VAR  VarI;


   for (i=0; i<BListSize (GnlFunctions (Gnl)); i++)
        {
         VarI = (GNL_VAR)BListElt (GnlFunctions (Gnl), i);
         FunctionI = GnlVarFunction (VarI);
         GnlResetNodeHook (GnlFunctionOnSet (FunctionI));
         GnlResetNodeHook (GnlFunctionDCSet (FunctionI));
        }
}

/*-------------------------------------------------------------------------*/
```

```
/* GnlNodeMaxDepth                                                    */
/*--------------------------------------------------------------------*/
/* Computes the Max depth from the node 'Node'. The Field 'Hook' is used*/
/* so it must be equals to 0 at the beginning if not a wrong max depth  */
/* can be returned.                                                   */
/*--------------------------------------------------------------------*/
int GnlNodeMaxDepth (Node)
   GNL_NODE Node;
{
   int            i;
   int            MaxDepth;
   int            DepthI;
   GNL_NODE SonI;
   GNL_NODE NodeFunction;
   GNL_VAR  Var;


   if (GnlNodeOp (Node) == GNL_CONSTANTE)
      return (0);

   if (GnlNodeOp (Node) == GNL_WIRE)
      {
       SonI = (GNL_NODE)GnlNodeSons (Node);
       return (GnlNodeMaxDepth (SonI));
      }

   if (GnlNodeOp (Node) == GNL_VARIABLE)
      {
       Var = (GNL_VAR)GnlNodeSons (Node);
       /*
       if (GnlVarDir (Var) != GNL_VAR_LOCAL)
          return (0);
        */

       if (GnlVarIsVss (Var))
          return (0);

       if (GnlVarIsVdd (Var))
          return (0);

       /* Continuing on the corresponding function.                */
       if (GnlVarFunction (Var) == NULL)
          {
           return (0);
          }
       /* we continue on the On set of the function.               */
       NodeFunction = GnlFunctionOnSet (GnlVarFunction (Var));
       return (GnlNodeMaxDepth (NodeFunction));
      }


  if (GnlNodeHook (Node))     /* different of NULL so we already got    */
                              /* the depth of this node previously.     */
     return ((int)GnlNodeHook (Node));

  MaxDepth = 0;
  for (i=0; i<BListSize (GnlNodeSons(Node)); i++)
```

```
        {
         SonI = (GNL_NODE)BListElt (GnlNodeSons(Node), i);
         DepthI = GnlNodeMaxDepth (SonI);
         if (DepthI > MaxDepth)
            MaxDepth = DepthI;
        }

   /* we store the result if we pass later on this node                */
   SetGnlNodeHook (Node, (int*)(MaxDepth+1));

   return (MaxDepth+1);
}


/*-------------------------------------------------------------------------*/
/* GnlGetGnlMaxDepth                                                       */
/*-------------------------------------------------------------------------*/
/* This procedures will use the field 'Hook' of each GNL_NODE so all the*/
/* informations in this field will be lost.                                */
/*-------------------------------------------------------------------------*/
int GnlGetGnlMaxDepth (Gnl)
   GNL      Gnl;
{
   int            i;
   int            MaxDepth;
   int            DepthI;
   GNL_FUNCTION   FunctionI;
   GNL_NODE NodeI;
   GNL_VAR  VarI;


   /* First we reset all the 'Hook' fields of each GNL_NODE object      */
   GnlResetGnlNodeHook (Gnl);

   MaxDepth = 0;
   for (i=0; i<BListSize (GnlFunctions (Gnl)); i++)
       {
        VarI = (GNL_VAR)BListElt (GnlFunctions (Gnl), i);
        FunctionI = GnlVarFunction (VarI);
      NodeI = GnlFunctionOnSet (FunctionI);

        DepthI = GnlNodeMaxDepth (NodeI);

        if (DepthI > MaxDepth)
           MaxDepth = DepthI;
       }

   /* First we reset all the 'Hook' fields of each GNL_NODE object      */
   /* to be clean at the output of this procedure                       */
   GnlResetGnlNodeHook (Gnl);

   return (MaxDepth);

}

/*-------------------------------------------------------------------------*/
/* GnlDepthComp                                                            */
```

```
/*-----------------------------------------------------------------------*/
static int GnlDepthComp (Var1, Var2)
    GNL_VAR *Var1;
    GNL_VAR *Var2;
{
    if ((*Var1)->Hook < (*Var2)->Hook)
       return (-1);

    if ((*Var1)->Hook == (*Var2)->Hook)
       return (0);

    return (1);
}


/*-----------------------------------------------------------------------*/
/* GnlDepthInvComp                                                       */
/*-----------------------------------------------------------------------*/
static int GnlDepthInvComp (Var1, Var2)
    GNL_VAR *Var1;
    GNL_VAR *Var2;
{
    if ((*Var1)->Hook < (*Var2)->Hook)
       return (1);

    if ((*Var1)->Hook == (*Var2)->Hook)
       return (0);

    return (-1);
}


/*-----------------------------------------------------------------------*/
/* GnlNodeLevel                                                          */
/*-----------------------------------------------------------------------*/
void GnlNodeLevel (Node, Level)
   GNL_NODE Node;
   int            Level;
{
   int            i;
   int            MaxDepth;
   int            DepthI;
   GNL_NODE SonI;
   GNL_NODE NodeFunction;
   GNL_VAR  Var;



   if (GnlNodeOp (Node) == GNL_VARIABLE)
      {
       Var = (GNL_VAR)GnlNodeSons (Node);
       if (GnlVarDir (Var) == GNL_VAR_INPUT)
          return ;
       if (GnlVarDir (Var) == GNL_VAR_INOUT)
          return ;

       /* If we passed previously with a higher level then it is not    */
```

```
        /* necessary to recurse.                                          */
        if (GnlVarHook (Var) >= (void*)Level)
         return;

        SetGnlVarHook (Var, (int*)Level);

        if (GnlVarIsVss (Var))
           return ;

        if (GnlVarIsVdd (Var))
           return ;

        /* Continuing on the corresponding function.                      */
        if (GnlVarFunction (Var) == NULL)
         return;

        /* we continue on the On set of the function.                     */
        NodeFunction = GnlFunctionOnSet (GnlVarFunction (Var));
        GnlNodeLevel (NodeFunction, Level+1);
        return;
       }

    if (GnlNodeOp (Node) == GNL_CONSTANTE)
       return ;

    for (i=0; i<BListSize (GnlNodeSons(Node)); i++)
        {
         SonI = (GNL_NODE)BListElt (GnlNodeSons(Node), i);
         GnlNodeLevel (SonI, Level);
        }

}


/*-------------------------------------------------------------------------*/
/* GnlSortFunctionsOnLevel                                                 */
/*-------------------------------------------------------------------------*/
/* This procedure sorts the Functions (GNL_VAR) according to their         */
/* Level in the netlist. The first function is the highest one and the     */
/* last is the deepest one.                                                */
/*-------------------------------------------------------------------------*/
void GnlSortFunctionsOnLevel (Gnl)
   GNL             Gnl;
{
   int             i;
   int             MaxDepth;
   int             DepthI;
   GNL_FUNCTION    FunctionI;
   GNL_VAR  VarI;
   BLIST    Functions;


   /* First we reset all the 'Hook' fields of each GNL_VAR object          */
   GnlResetVarHook (Gnl);

   for (i=0; i<BListSize (GnlFunctions (Gnl)); i++)
       {
```

```
        VarI = (GNL_VAR)BListElt (GnlFunctions (Gnl), i);
        FunctionI = GnlVarFunction (VarI);
        GnlNodeLevel (GnlFunctionOnSet (FunctionI), 1);
       }

   Functions = GnlFunctions (Gnl);
   qsort (BListAdress (Functions), BListSize (Functions),
          sizeof(GNL_VAR), GnlDepthComp);

   /* Finally we reset all the 'Hook' fields of each GNL_VAR object     */
   GnlResetVarHook (Gnl);
}

/*-------------------------------------------------------------------------*/
/* GnlSortFunctionsOnInvertLevel                                           */
/*-------------------------------------------------------------------------*/
/* This procedure sorts the Functions (GNL_VAR) according to their         */
/* Level in the netlist. The first function is the deepest one and the     */
/* last is the highets one.                                                */
/*-------------------------------------------------------------------------*/
void GnlSortFunctionsOnInvertLevel (Gnl)
   GNL            Gnl;
{
   int            i;
   int            MaxDepth;
   int            DepthI;
   GNL_FUNCTION   FunctionI;
   GNL_VAR  VarI;
   BLIST    Functions;


   /* First we reset all the 'Hook' fields of each GNL_VAR object      */
   GnlResetVarHook (Gnl);

   for (i=0; i<BListSize (GnlFunctions (Gnl)); i++)
       {
        VarI = (GNL_VAR)BListElt (GnlFunctions (Gnl), i);
        FunctionI = GnlVarFunction (VarI);
        GnlNodeLevel (GnlFunctionOnSet (FunctionI), 1);
       }

   Functions = GnlFunctions (Gnl);
   qsort (BListAdress (Functions), BListSize (Functions),
          sizeof(GNL_VAR), GnlDepthInvComp);

   /* Finally we reset all the 'Hook' fields of each GNL_VAR object     */
   GnlResetVarHook (Gnl);
}

/*-------------------------------------------------------------------------*/
/* GnlSortFunctionsOnInvertLevelStoreLevel                                 */
/*-------------------------------------------------------------------------*/
/* This procedure sorts the Functions (GNL_VAR) according to their         */
/* Level in the netlist. The first function is the deepest one and the     */
/* last is the highest one.                                                */
/*-------------------------------------------------------------------------*/
void GnlSortFunctionsOnInvertLevelStoreLevel (Gnl)
```

```
   GNL            Gnl;
{
   int            i;
   int            MaxDepth;
   int            DepthI;
   GNL_FUNCTION   FunctionI;
   GNL_VAR  VarI;
   BLIST    Functions;


   /* First we reset all the 'Hook' fields of each GNL_VAR object      */
   GnlResetVarHook (Gnl);

   for (i=0; i<BListSize (GnlFunctions (Gnl)); i++)
       {
        VarI = (GNL_VAR)BListElt (GnlFunctions (Gnl), i);
        FunctionI = GnlVarFunction (VarI);
        GnlNodeLevel (GnlFunctionOnSet (FunctionI), 1);
       }

   Functions = GnlFunctions (Gnl);
   qsort (BListAdress (Functions), BListSize (Functions),
          sizeof(GNL_VAR), GnlDepthInvComp);

}

/*-------------------------------------------------------------------------*/
/* GnlGetGnlInfo                                                           */
/*-------------------------------------------------------------------------*/
/* This procedure computes different kind of information about the      */
/* current Gnl like: Number Flip-Flops, Number Latches, Number Literals */
/* Maximum combinatorial critcal path, ...                              */
/* If returns a structure thru 'GnlInfo' storing all these information  */
/*-------------------------------------------------------------------------*/
GNL_STATUS GnlGetGnlInfo (Gnl, GnlInfo)
   GNL            Gnl;
   GNL_INFO *GnlInfo;
{
   int            NbFlipsFlops;
   int            NbLatches;
   int            NbTriStates;
   int            NbBuffers;
   int            NbLiterals;
   float    NbCombEquiGates;
   float    NbSeqEquiGates;
   float    NbTotalEquiGates;
   int            MaxDepth;
   int            i;
   int            NbInputs;
   int            NbOutputs;
   int            NbInOuts;
   GNL_VAR  PortI;


   if ((*GnlInfo = (GNL_INFO)calloc (1, sizeof (GNL_INFO_REC))) == NULL)
      return (GNL_MEMORY_FULL);
```

```
    NbFlipsFlops = GnlGetNbFlipsFlops (Gnl);
    NbLatches = GnlGetNbLatches (Gnl);
    NbTriStates = GnlGetNbTriStates (Gnl);
    NbBuffers = GnlGetNbBuffers (Gnl);
    NbLiterals = GnlGetGnlNbLiterals (Gnl);
    GnlGetGnlNbEquiGates (Gnl, &NbCombEquiGates, &NbSeqEquiGates);
    NbTotalEquiGates = NbCombEquiGates+NbSeqEquiGates;
    MaxDepth = GnlGetGnlMaxDepth (Gnl);

    NbInputs = NbOutputs = NbInOuts = 0;
    for (i=0; i<BListSize (GnlListPorts (Gnl)); i++)
        {
         PortI = (GNL_VAR)BListElt (GnlListPorts (Gnl), i);
         switch (GnlVarDir (PortI)) {
             case GNL_VAR_INPUT:
                      NbInputs++;
                      break;

             case GNL_VAR_OUTPUT:
                      NbOutputs++;
                      break;

             case GNL_VAR_INOUT:
                      NbInOuts++;
                      break;
                }
        }

    SetGnlInfoNbInputs (*GnlInfo, NbInputs);
    SetGnlInfoNbOutputs (*GnlInfo, NbOutputs);
    SetGnlInfoNbInOuts (*GnlInfo, NbInOuts);

    SetGnlInfoNbClocks (*GnlInfo, BListSize (GnlClocks (Gnl)));
    SetGnlInfoNbFlipFlops (*GnlInfo, NbFlipsFlops);
    SetGnlInfoNbLatches (*GnlInfo, NbLatches);
    SetGnlInfoNbTriStates (*GnlInfo, NbTriStates);
    SetGnlInfoNbBuffers (*GnlInfo, NbBuffers);

    SetGnlInfoNbLit (*GnlInfo, NbLiterals);
    SetGnlInfoNbCombEquiGates (*GnlInfo, NbCombEquiGates);
    SetGnlInfoNbSeqEquiGates (*GnlInfo, NbSeqEquiGates);
    SetGnlInfoNbTotalEquiGates (*GnlInfo, NbTotalEquiGates);
    SetGnlInfoMaxDepth (*GnlInfo, MaxDepth);

    return (GNL_OK);
}

/*-------------------------------------------------------------------------*/
/* GnlPrintGnlInfo                                                         */
/*-------------------------------------------------------------------------*/
GNL_STATUS GnlPrintGnlInfo (File, Gnl)
    FILE           *File;
    GNL            Gnl;
{
    GNL_INFO GnlInfo;
```

```
   if (GnlGetGnlInfo (Gnl, &GnlInfo))
      return (GNL_MEMORY_FULL);

   fprintf (File, "# -------------------------------\n");
   fprintf (File, "# | INPUTS            =%9d |\n",
            GnlNbIn (Gnl));
   fprintf (File, "# | OUTPUTS           =%9d |\n",
            GnlNbOut (Gnl));
   fprintf (File, "# | LOCALS            =%9d |\n",
            GnlNbLocal (Gnl));
   fprintf (File, "# | CLOCKS            =%9d |\n",
            GnlInfoNbClocks (GnlInfo));
   fprintf (File, "# | FLIP-FLOPS        =%9d |\n",
            GnlInfoNbFlipFlops (GnlInfo));
   fprintf (File, "# | LATCHES           =%9d |\n",
            GnlInfoNbLatches (GnlInfo));
   fprintf (File, "# | TRISTATES         =%9d |\n",
            GnlInfoNbTriStates (GnlInfo));
   fprintf (File, "# | BUFFERS           =%9d |\n",
            GnlInfoNbBuffers (GnlInfo));
   fprintf (File, "# | LITERALS          =%9d |\n",
            GnlInfoNbLit (GnlInfo));
   /*
   fprintf (File, "# | COMB.EQUIV.GATES  =%9.1f |\n",
         GnlInfoNbCombEquiGates (GnlInfo));
   fprintf (File, "# | SEQ.EQUIV.GATES   =%9.1f |\n",
         GnlInfoNbSeqEquiGates (GnlInfo));
   fprintf (File, "# | TOTAL EQUIV.GATES =%9.1f |\n",
         GnlInfoNbTotalEquiGates (GnlInfo));
         */
   fprintf (File, "# | COMB.DEPTH        =%9d |\n",
            GnlInfoMaxDepth (GnlInfo));
   fprintf (File, "# -------------------------------\n");

   free ((char*)GnlInfo);

   return (GNL_OK);
}

/*-------------------------------------------------------------------------*/
/* GnlGnlInfoPrint                                                         */
/*-------------------------------------------------------------------------*/
void GnlGnlInfoPrint (File, Gnl, GnlInfo)
   FILE          *File;
   GNL           Gnl;
   GNL_INFO GnlInfo;
{

   fprintf (File, "# -------------------------------\n");
   fprintf (File, "# | INPUTS            =%9d |\n",
            GnlNbIn (Gnl));
   fprintf (File, "# | OUTPUTS           =%9d |\n",
            GnlNbOut (Gnl));
   fprintf (File, "# | LOCALS            =%9d |\n",
            GnlNbLocal (Gnl));
   fprintf (File, "# | CLOCKS            =%9d |\n",
            GnlInfoNbClocks (GnlInfo));
```

```
   fprintf (File, "# | FLIP-FLOPS        =%9d |\n",
            GnlInfoNbFlipFlops (GnlInfo));
   fprintf (File, "# | LATCHES           =%9d |\n",
            GnlInfoNbLatches (GnlInfo));
   fprintf (File, "# | TRISTATES         =%9d |\n",
            GnlInfoNbTriStates (GnlInfo));
   fprintf (File, "# | BUFFERS           =%9d |\n",
            GnlInfoNbBuffers (GnlInfo));
   fprintf (File, "# | LITERALS          =%9d |\n",
            GnlInfoNbLit (GnlInfo));
   /*
   fprintf (File, "# | COMB.EQUIV.GATES  =%9.1f |\n",
          GnlInfoNbCombEquiGates (GnlInfo));
   fprintf (File, "# | SEQ.EQUIV.GATES   =%9.1f |\n",
          GnlInfoNbSeqEquiGates (GnlInfo));
   fprintf (File, "# | TOTAL EQUIV.GATES =%9.1f |\n",
          GnlInfoNbTotalEquiGates (GnlInfo));
          */
   fprintf (File, "# | COMB.DEPTH        =%9d |\n",
            GnlInfoMaxDepth (GnlInfo));
   fprintf (File, "# -------------------------------\n");

   free ((char*)GnlInfo);

}

/*-------------------------------------------------------------------------*/
/* GnlFreeNodeSons                                                         */
/*-------------------------------------------------------------------------*/
void GnlFreeNodeSons (Node)
   GNL_NODE Node;
{
   int            i;
   GNL_NODE SonI;


   if (Node == NULL)
      return;

   if (GnlNodeOp (Node) == GNL_CONSTANTE)
      return;

   if (GnlNodeOp (Node) == GNL_VARIABLE)
      return;

   if (GnlNodeOp (Node) == GNL_WIRE)
      return;

   if (GnlNodeSons (Node) == NULL)
      return;

   for (i=0; i <BListSize (GnlNodeSons (Node)); i++)
       {
      SonI = (GNL_NODE)BListElt (GnlNodeSons (Node), i);
      GnlFreeNodeSons (SonI);
       }
   BListQuickDelete (&GnlNodeSons (Node));
```

```
}


/*-------------------------------------------------------------------------*/
/* GnlFreeHashNames                                                        */
/*-------------------------------------------------------------------------*/
/* Physical destruction the Hash Table of Gnl Variables in the Gnl 'Gnl'*/
/*-------------------------------------------------------------------------*/
void GnlFreeHashNames (Gnl)
   GNL      Gnl;
{
   BLIST    HashTableNames;
   int            i;
   BLIST    BucketI;
   int            j;
   GNL_VAR  VarJ;


   HashTableNames = GnlHashNames (Gnl);

   for (i=0; i<BListSize (HashTableNames); i++)
       {
      BucketI = (BLIST)BListElt (HashTableNames, i);
      for (j=0; j<BListSize (BucketI); j++)
            {
             VarJ = (GNL_VAR)BListElt (BucketI, j);
           if (GnlVarIsVdd (VarJ) || GnlVarIsVss (VarJ))
            continue;
           GnlFreeVar (VarJ);
          }
      if (BucketI)
         BListQuickDelete (&BucketI);
       }

   BListQuickDelete (&HashTableNames);

   SetGnlHashNames (Gnl, NULL);
}

/*-------------------------------------------------------------------------*/
/* GnlFreeNodesSegments                                                    */
/*-------------------------------------------------------------------------*/
/* Physical destruction of the segments storing all the Gnl Nodes which */
/* have been created in the gnl 'Gnl'.                                     */
/*-------------------------------------------------------------------------*/
void GnlFreeNodesSegments (Gnl)
   GNL      Gnl;
{
   int            i;
   int            j;
   GNL_NODE Node;
   BLIST    ListNodesSegments;
   GNL_NODE SegmentI;
   GNL_VAR  VarI;
   GNL_FUNCTION   FunctionI;
```

```
    for (i=0; i<BListSize (GnlFunctions (Gnl)); i++)
        {
         VarI = (GNL_VAR)BListElt (GnlFunctions (Gnl), i);
         FunctionI = GnlVarFunction (VarI);
       GnlFreeNodeSons (GnlFunctionOnSet (FunctionI));
       GnlFreeNodeSons (GnlFunctionDCSet (FunctionI));
        }

    ListNodesSegments = GnlNodesSegments (Gnl);

    for (i=0; i<BListSize (ListNodesSegments)-1; i++)
        {
       SegmentI = (GNL_NODE)BListElt (ListNodesSegments, i);
         Node = SegmentI;
         for (j=0; j<SEGMENT_NODE_SIZE; j++)
             {
              if (GnlNodeOp (Node) == GNL_CONSTANTE)
                 continue;
              if (GnlNodeOp (Node) == GNL_VARIABLE)
                 continue;
              BListQuickDelete (&GnlNodeSons (Node));
              SetGnlNodeSons (Node, NULL);
              Node++;
             }

       free ((char*)SegmentI);
        }

    SegmentI = (GNL_NODE)BListElt (ListNodesSegments, i);
    Node = SegmentI;
    while (Node < GnlLastNode (Gnl))
          {
           if (GnlNodeOp (Node) == GNL_CONSTANTE)
              {
               Node++;
               continue;
              }
           if (GnlNodeOp (Node) == GNL_VARIABLE)
              {
               Node++;
               continue;
              }
           BListQuickDelete (&GnlNodeSons (Node));
           SetGnlNodeSons (Node, NULL);
           Node++;
          }
    free ((char*)SegmentI);

    SetGnlFirstNode (Gnl, NULL);
    SetGnlLastNode (Gnl, NULL);

}

/*-------------------------------------------------------------------------*/
/* GnlPropagateInvInNode                                                   */
/*-------------------------------------------------------------------------*/
GNL_STATUS GnlPropagateInvInNode (Gnl, Node, DoNegation, NewNode)
```

```
   GNL            Gnl;
   GNL_NODE Node;
   int            DoNegation;
   GNL_NODE *NewNode;
{
   int            i;
   BLIST    SonList;
   GNL_NODE SonI;
   BLIST    Sons;
   GNL_NODE SNode;
   GNL_NODE VarNode;


   if (Node == NULL)
      {
       *NewNode = NULL;
       return (GNL_OK);
      }

   if (GnlCreateNode (Gnl, NULL, NewNode))
      return (GNL_MEMORY_FULL);

   switch (GnlNodeOp (Node)) {
      case GNL_AND:
           if (BListCreateWithSize (1, &SonList))
                return (GNL_MEMORY_FULL);
             if (!DoNegation)
                {
             SetGnlNodeOp (*NewNode, GNL_AND);
             Sons = GnlNodeSons (Node);
             for (i=0; i<BListSize (Sons); i++)
                    {
                     SonI = (GNL_NODE)BListElt (Sons, i);
                  if (GnlPropagateInvInNode (Gnl, SonI, 0, &SNode))
                         return (GNL_MEMORY_FULL);
                  if (BListAddElt (SonList, (int)SNode))
                         return (GNL_MEMORY_FULL);
                    }
             SetGnlNodeSons (*NewNode, SonList);
             return (GNL_OK);
            }
           else
            {
             SetGnlNodeOp (*NewNode, GNL_OR);
             Sons = GnlNodeSons (Node);
             for (i=0; i<BListSize (Sons); i++)
                 {
                  SonI = (GNL_NODE)BListElt (Sons, i);
                  if (GnlPropagateInvInNode (Gnl, SonI, 1, &SNode))
                     return (GNL_MEMORY_FULL);
                    if (BListAddElt (SonList, (int)SNode))
                       return (GNL_MEMORY_FULL);
                   }
               SetGnlNodeSons (*NewNode, SonList);
               return (GNL_OK);
              }
          break;
```

```
case GNL_OR:
     if (BListCreateWithSize (1, &SonList))
          return (GNL_MEMORY_FULL);
       if (!DoNegation)
          {
       SetGnlNodeOp (*NewNode, GNL_OR);
       Sons = GnlNodeSons (Node);
       for (i=0; i<BListSize (Sons); i++)
              {
               SonI = (GNL_NODE)BListElt (Sons, i);
            if (GnlPropagateInvInNode (Gnl, SonI, 0, &SNode))
                  return (GNL_MEMORY_FULL);
            if (BListAddElt (SonList, (int)SNode))
                  return (GNL_MEMORY_FULL);
           }
       SetGnlNodeSons (*NewNode, SonList);
       return (GNL_OK);
      }
     else
      {
       SetGnlNodeOp (*NewNode, GNL_AND);
       Sons = GnlNodeSons (Node);
       for (i=0; i<BListSize (Sons); i++)
           {
            SonI = (GNL_NODE)BListElt (Sons, i);
            if (GnlPropagateInvInNode (Gnl, SonI, 1, &SNode))
               return (GNL_MEMORY_FULL);
             if (BListAddElt (SonList, (int)SNode))
                return (GNL_MEMORY_FULL);
             }
         SetGnlNodeSons (*NewNode, SonList);
         return (GNL_OK);
        }
     break;


case GNL_NOT:
       Sons = GnlNodeSons (Node);
       SonI = (GNL_NODE)BListElt (Sons, 0);
     if (DoNegation)
        return (GnlPropagateInvInNode (Gnl, SonI, 0, NewNode));
     else
        return (GnlPropagateInvInNode (Gnl, SonI, 1, NewNode));
       break;


case GNL_VARIABLE:
       if (!DoNegation)
          {
           SetGnlNodeOp (*NewNode, GNL_VARIABLE);
       SetGnlNodeSons (*NewNode, GnlNodeSons (Node));
       return (GNL_OK);
      }
     else
      {
```

```
            SetGnlNodeOp (*NewNode, GNL_NOT);
            if (BListCreateWithSize (1, &SonList))
               return (GNL_MEMORY_FULL);

                if (GnlCreateNode (Gnl, GNL_VARIABLE, &VarNode))
                   return (GNL_MEMORY_FULL);
            SetGnlNodeSons (VarNode, GnlNodeSons (Node));
            if (BListAddElt (SonList, (int)VarNode))
                   return (GNL_MEMORY_FULL);
            SetGnlNodeSons (*NewNode, SonList);
            return (GNL_OK);
           }
         break;


      case GNL_CONSTANTE:
           SetGnlNodeOp (*NewNode, GNL_CONSTANTE);
           if (DoNegation)
            SetGnlNodeSons (*NewNode, (BLIST)!GnlNodeSons(Node));
           else
            SetGnlNodeSons (*NewNode, (BLIST)GnlNodeSons(Node));
           return (GNL_OK);

      }

}

/*-------------------------------------------------------------------------*/
/* GnlPropagateInv                                                         */
/*-------------------------------------------------------------------------*/
/* This procedure propagates the GNL_NOT node operator down to the         */
/* leaves. Nodes trees are physically duplicated and new segments of       */
/* nodes are created in 'Gnl'. The original segments of nodes of 'Gnl'     */
/* are freed.                                                              */
/*-------------------------------------------------------------------------*/
GNL_STATUS GnlPropagateInv (Gnl)
   GNL            Gnl;
{
   int            i;
   char           *CopyName;
   GNL_VAR  VarI;
   GNL_FUNCTION   FunctionI;
   GNL_NODE NewOnSet;
   GNL_NODE NewDCSet;
   BLIST    NewListFunctions;
   int            j;
   BLIST    NodesSegments;
   GNL            NewGnl;


   /* We use an intermediate GNL to do the propagation and duplication  */
   if ((NewGnl = (GNL)calloc (1, sizeof(GNL_REC))) == NULL)
      return (GNL_MEMORY_FULL);

   SetGnlNbIn (NewGnl, GnlNbIn (Gnl));
   SetGnlNbOut (NewGnl, GnlNbOut (Gnl));
   SetGnlNbLocal (NewGnl, GnlNbLocal (Gnl));
```

```
   SetGnlInputs (NewGnl, GnlInputs (Gnl));
   SetGnlOutputs (NewGnl, GnlOutputs(Gnl));
   SetGnlLocals (NewGnl, GnlLocals (Gnl));

   SetGnlHashNames (NewGnl, GnlHashNames (Gnl));

   if (BListCreate (&NodesSegments))
      return (GNL_MEMORY_FULL);
   SetGnlNodesSegments (NewGnl, NodesSegments);
   SetGnlFirstNode (NewGnl, NULL);
   SetGnlLastNode (NewGnl, NULL);

   SetGnlComponents (NewGnl, GnlComponents (Gnl));

   if (BListCreate (&NewListFunctions))
      return (GNL_MEMORY_FULL);

   for (i=0; i<BListSize (GnlFunctions (Gnl)); i++)
       {
        VarI = (GNL_VAR)BListElt (GnlFunctions (Gnl), i);
      FunctionI = GnlVarFunction (VarI);

      if (GnlPropagateInvInNode (NewGnl, GnlFunctionOnSet (FunctionI),
                                 0, &NewOnSet))
         return (GNL_MEMORY_FULL);

      if (GnlPropagateInvInNode (NewGnl, GnlFunctionDCSet (FunctionI),
                                 0, &NewDCSet))
         return (GNL_MEMORY_FULL);

      /* 'NewListFunctions' is a list of couples: NewOnSet node and     */
      /* 'NewDCSet' node.                                               */
        if (BListAddElt (NewListFunctions, (int)NewOnSet))
         return (GNL_MEMORY_FULL);

        if (BListAddElt (NewListFunctions, (int)NewDCSet))
         return (GNL_MEMORY_FULL);
       }

   GnlFreeNodesSegments (Gnl);

   i = 0;
   for (j=0; j<BListSize (NewListFunctions)-1; j += 2)
       {
        VarI = (GNL_VAR)BListElt (GnlFunctions (Gnl), i);
      FunctionI = GnlVarFunction (VarI);

      NewOnSet = (GNL_NODE)BListElt (NewListFunctions, j);
      NewDCSet = (GNL_NODE)BListElt (NewListFunctions, j+1);

      /* we update the function 'FunctionI' with the new Nodes.   */
      SetGnlFunctionOnSet (FunctionI, NewOnSet);
      SetGnlFunctionDCSet (FunctionI, NewDCSet);

      i++;
       }
```

```
   BListQuickDelete (&NewListFunctions);

   /* We move the segments of nodes from 'NewGnl' to 'Gnl'.       */
   SetGnlNodesSegments (Gnl, GnlNodesSegments (NewGnl));
   SetGnlFirstNode (Gnl, GnlFirstNode (NewGnl));
   SetGnlLastNode (Gnl, GnlLastNode (NewGnl));

   free ((char*)NewGnl);

   return (GNL_OK);
}

/*-------------------------------------------------------------------------*/
/* GnlFreeConcatNode                                                       */
/*-------------------------------------------------------------------------*/
void GnlFreeConcatNode (ConcatNode)
   GNL_NODE ConcatNode;
{

   BListQuickDelete (&(GnlNodeSons (ConcatNode)));
   free (ConcatNode);
}

/*-------------------------------------------------------------------------*/
/* GnlFreeAssoc                                                            */
/*-------------------------------------------------------------------------*/
void GnlFreeAssoc (Assoc)
   GNL_ASSOC      Assoc;
{
   GNL_VAR  Actual;

   /* Formal is a GNL_VAR and has been freed in the Hash names freeing  */
   Actual = GnlAssocActualPort (Assoc);

   if (!Actual || GnlVarIsVar (Actual))
      {
       free (Assoc);
       return;
      }

   /* Actual is a GNL_NODE (GNL_CONCAT) that we can free          */
   GnlFreeConcatNode (Actual);

   free (Assoc);
}

/*-------------------------------------------------------------------------*/
/* GnlFreeUserComponent                                                    */
/*-------------------------------------------------------------------------*/
void GnlFreeUserComponent (UserCompo)
   GNL_USER_COMPONENT   UserCompo;
{
   BLIST    Interface;
   int            i;
   GNL_ASSOC      AssocI;
```

```
    Interface = GnlUserComponentInterface (UserCompo);
    for (i=0; i<BListSize (Interface); i++)
        {
       AssocI = (GNL_ASSOC)BListElt (Interface, i);
       GnlFreeAssoc (AssocI);
        }

    BListQuickDelete (&Interface);

    free (GnlUserComponentInstName (UserCompo));

    free (UserCompo);
}

/*-------------------------------------------------------------------------*/
/* GnlFreeBufComponent                                                     */
/*-------------------------------------------------------------------------*/
void GnlFreeBufComponent (BufCompo)
    GNL_BUF_COMPONENT   BufCompo;
{
    BLIST         Interface;
    int           i;
    GNL_ASSOC     AssocI;


    Interface = GnlBufInterface (BufCompo);
    for (i=0; i<BListSize (Interface); i++)
        {
         AssocI = (GNL_ASSOC)BListElt (Interface, i);
         GnlFreeAssoc (AssocI);
        }

    BListQuickDelete (&Interface);

    if (GnlBufInstName (BufCompo))
       free (GnlBufInstName (BufCompo));

    free (BufCompo);
}


/*-------------------------------------------------------------------------*/
/* GnlFreeComponent                                                        */
/*-------------------------------------------------------------------------*/
void GnlFreeComponent (Component)
    GNL_COMPONENT  Component;
{
    GNL_USER_COMPONENT   UserCompo;

    switch (GnlComponentType (Component)) {
       case GNL_SEQUENTIAL_COMPO:
       case GNL_TRISTATE_COMPO:
       case GNL_BUF_COMPO:
       case GNL_MACRO_COMPO:
            return;

       case GNL_USER_COMPO:
```

```
            UserCompo = (GNL_USER_COMPONENT)Component;
            GnlFreeUserComponent (UserCompo);
            return;
        }
}

/*-------------------------------------------------------------------------*/
/* GnlUpdateComponentGnlDefOfGnl                                           */
/*-------------------------------------------------------------------------*/
void GnlUpdateComponentGnlDefOfGnl (Gnl, ListGnls)
   GNL             Gnl;
   BLIST         ListGnls;
{
   int                   i;
   GNL_COMPONENT   ComponentI;
   GNL_USER_COMPONENT    UserCompo;
   int                   j;
   GNL                   GnlJ;


   for (i=0; i<BListSize (GnlComponents (Gnl)); i++)
        {
         ComponentI = (GNL_COMPONENT)BListElt (GnlComponents (Gnl), i);
         if (GnlComponentType (ComponentI) != GNL_USER_COMPO)
            continue;
         UserCompo = (GNL_USER_COMPONENT)ComponentI;

         for (j=0; j<BListSize (ListGnls); j++)
              {
               GnlJ = (GNL)BListElt (ListGnls, j);
               if (!strcmp (GnlName (GnlJ),
                            GnlUserComponentName (UserCompo)))
                  {
                   SetGnlUserComponentGnlDef (UserCompo, GnlJ);
                   break;
                  }
              }
        }
}


/*-------------------------------------------------------------------------*/
/* GnlUpdateComponentGnlDef                                                */
/*-------------------------------------------------------------------------*/
void GnlUpdateComponentGnlDef (ListGnls)
   BLIST    ListGnls;
{
   int             i;
   GNL             GnlI;


   for (i=0; i<BListSize (ListGnls); i++)
        {
         GnlI = (GNL)BListElt (ListGnls, i);
         GnlUpdateComponentGnlDefOfGnl (GnlI, ListGnls);
        }
}
```

```
/*-------------------------------------------------------------------------*/
/* GnlUpdateFanoutOfVar                                                    */
/*-------------------------------------------------------------------------*/
void GnlUpdateFanoutOfVar (Var)
   GNL_VAR  Var;

{
   GNL_NODE Node;
   GNL_VAR  NewVar;


   if (GnlVarIsVss (Var) || GnlVarIsVdd (Var))
      return;

   /* 'Var' is then used one more time                                     */
   SetGnlVarHook (Var, (int)GnlVarHook (Var)+1);

}

/*-------------------------------------------------------------------------*/
/* GnlUpdateFanoutOfActual                                                 */
/*-------------------------------------------------------------------------*/
void GnlUpdateFanoutOfActual (Actual)
   GNL_VAR  Actual;
{
   int                  i;
   BLIST          Sons;
   GNL_NODE       NodeI;
   GNL_VAR        VarI;


   if (GnlVarIsVar (Actual))
      {
       GnlUpdateFanoutOfVar (Actual);
       return;
      }

   /* The actual is a GNL_CONCAT node so all the sons are GNL_VARIABLE  */
   /* nodes.                                                            */
   Sons = GnlNodeSons ((GNL_NODE)Actual);
   for (i=0; i<BListSize (Sons); i++)
       {
        VarI = (GNL_VAR)BListElt (Sons, i);
        GnlUpdateFanoutOfVar (VarI);
       }

}


/*-------------------------------------------------------------------------*/
/* GnlUpdateFanoutFromUserCompo                                            */
/*-------------------------------------------------------------------------*/
void GnlUpdateFanoutFromUserCompo (UserCompo)
   GNL_USER_COMPONENT   UserCompo;
{
   int            i;
```

```
    GNL_ASSOC      AssocI;
    GNL_VAR  Actual;
    GNL_VAR  Formal;
    BLIST    Interface;


    Interface = GnlUserComponentInterface (UserCompo);

    for (i=0; i<BListSize (Interface); i++)
        {
         AssocI = (GNL_ASSOC)BListElt (Interface, i);
         Actual = GnlAssocActualPort (AssocI);

         if (!Actual)
            continue;

         Formal = GnlAssocFormalPort (AssocI);

         if (GnlVarDir (Formal) == GNL_VAR_OUTPUT)
            continue;

         GnlUpdateFanoutOfActual (Actual);
        }
}

/*-------------------------------------------------------------------------*/
/* GnlUpdateFanoutFromUserCompoCell                                        */
/*-------------------------------------------------------------------------*/
void GnlUpdateFanoutFromUserCompoCell (UserCompo)
    GNL_USER_COMPONENT   UserCompo;
{
    int             i;
    GNL_ASSOC              AssocI;
    GNL_VAR                Actual;
    char                   *Formal;
    BLIST                  Interface;
    LIBC_PIN       PinFormal;
    LIBC_CELL            Cell;
    LIBC_NAME_LIST ListPinName;


    Interface = GnlUserComponentInterface (UserCompo);
    Cell = GnlUserComponentCellDef (UserCompo);

    /* If it is a real black box we do not count the ports         */
    if (!Cell)
       return ;

    for (i=0; i<BListSize (Interface); i++)
        {
         AssocI = (GNL_ASSOC)BListElt (Interface, i);
         Actual = GnlAssocActualPort (AssocI);

         if (!Actual)
            continue;

         Formal = GnlAssocFormalPort (AssocI);
```

```
        PinFormal = GnlGetPinCellWithName (Cell, Formal);

        if (!PinFormal)
           {
            fprintf (stderr,
                     " ERROR: cannot find pin name <%s> in cell [%s]\n",
                     Formal, LibCellName (Cell));
            exit (1);
           }

        if (LibPinDirection (PinFormal) == OUTPUT_E)
           continue;

        GnlUpdateFanoutOfActual (Actual);
       }

}

/*-------------------------------------------------------------------------*/
/* GnlUpdateFanoutFromUserCompoKeepVar                                     */
/*-------------------------------------------------------------------------*/
void GnlUpdateFanoutFromUserCompoKeepVar (UserCompo)
   GNL_USER_COMPONENT   UserCompo;
{
   int          i;
   GNL_ASSOC    AssocI;
   GNL_VAR      Actual;
   char         *Formal;
   BLIST        Interface;
   LIBC_PIN PinFormal;
   LIBC_CELL      Cell;


   Interface = GnlUserComponentInterface (UserCompo);
   Cell = GnlUserComponentCellDef (UserCompo);

   /* If it is a real black box we do not count the ports         */
   if (!Cell)
      return;

   for (i=0; i<BListSize (Interface); i++)
       {
        AssocI = (GNL_ASSOC)BListElt (Interface, i);
        Actual = GnlAssocActualPort (AssocI);

        if (!Actual)
           continue;

        Formal = GnlAssocFormalPort (AssocI);
        PinFormal = GnlGetPinCellWithName (Cell, Formal);

        if (!PinFormal)
           {
     fprintf (stderr, " ERROR: cannot find pin name <%s> in cell [%s]\n",
                      Formal, LibCellName (Cell));
            exit (1);
           }
```

```
        if (LibPinDirection (PinFormal) == OUTPUT_E)
           continue;

        GnlUpdateFanoutOfActual (Actual);
       }
}

/*------------------------------------------------------------------------*/
/* GnlUpdateFanoutFromSeqCompo                                            */
/*------------------------------------------------------------------------*/
void GnlUpdateFanoutFromSeqCompo (SeqCompo)
   GNL_SEQUENTIAL_COMPONENT   SeqCompo;
{
   GNL_VAR  Signal;

   Signal = GnlSequentialCompoInput (SeqCompo);
   GnlUpdateFanoutOfActual (Signal);

   Signal = GnlSequentialCompoClock (SeqCompo);
   GnlUpdateFanoutOfActual (Signal);

   Signal = GnlSequentialCompoReset (SeqCompo);
   if (Signal)
      GnlUpdateFanoutOfActual (Signal);

   Signal = GnlSequentialCompoSet (SeqCompo);
   if (Signal)
      GnlUpdateFanoutOfActual (Signal);

}

/*------------------------------------------------------------------------*/
/* GnlUpdateFanoutFromTriStateCompo                                       */
/*------------------------------------------------------------------------*/
void GnlUpdateFanoutFromTriStateCompo (TriStateCompo)
   GNL_TRISTATE_COMPONENT     TriStateCompo;
{
   GNL_VAR      Signal;


   Signal = GnlTriStateInput (TriStateCompo);
   GnlUpdateFanoutOfActual (Signal);

   Signal = GnlTriStateSelect (TriStateCompo);
   GnlUpdateFanoutOfActual (Signal);

}

/*------------------------------------------------------------------------*/
/* GnlUpdateFanoutFromBufCompo                                            */
/*------------------------------------------------------------------------*/
void GnlUpdateFanoutFromBufCompo (BufCompo)
   GNL_BUF_COMPONENT     BufCompo;
{
   GNL_VAR      Signal;
```

```
   Signal = GnlBufInput (BufCompo);
   GnlUpdateFanoutOfActual (Signal);

}

/*-------------------------------------------------------------------------*/
/* GnlUpdateFanoutOnVarRec                                                 */
/*-------------------------------------------------------------------------*/
void GnlUpdateFanoutOnVarRec (Var)
   GNL_VAR  Var;
{
   GNL_NODE Node;
   GNL_VAR  RecVar;


   GnlUpdateFanoutOfVar (Var);

   if (!GnlVarFunction (Var))
      return;

   Node = GnlFunctionOnSet (GnlVarFunction (Var));
   if (GnlNodeOp (Node) != GNL_VARIABLE)
      return;

   RecVar = (GNL_VAR)GnlNodeSons (Node);

   GnlUpdateFanoutOnVarRec (RecVar);
}

/*-------------------------------------------------------------------------*/
/* GnlComputeMaxFanoutInGnl                                                */
/*-------------------------------------------------------------------------*/
/* This procedure computes the MaxFanout found in the gnl 'Gnl' and        */
/* stores it in the field 'GnlMaxFanout (Gnl)'.                            */
/* We use the field 'GnlVarHook' of each var to store the number of        */
/* times this variable is used.                                            */
/* The GNL must be composed only of components and simple assignements     */
/*-------------------------------------------------------------------------*/
void GnlComputeMaxFanoutInGnl (Gnl)
   GNL            Gnl;
{
   int                        i;
   GNL_COMPONENT        ComponentI;
   int                        MaxFanout;
   BLIST                BucketI;
   int                        j;
   GNL_VAR              VarJ;
   BLIST                HashTableNames;
   BLIST                Interface;
   GNL_USER_COMPONENT         UserCompo;
   GNL_TRISTATE_COMPONENT     TristateCompo;
   GNL_BUF_COMPONENT          BufCompo;
   GNL_SEQUENTIAL_COMPONENT   SeqCompo;
   GNL_VAR              VarI;
```

```
    /* We reset the Hook filed of all the Vars of the Gnl.          */
    GnlResetVarHook (Gnl);

    for (i=0; i<BListSize (GnlComponents (Gnl)); i++)
        {
         ComponentI = (GNL_COMPONENT)BListElt (GnlComponents (Gnl), i);
         switch (GnlComponentType (ComponentI)) {
             case GNL_USER_COMPO:
                      UserCompo = (GNL_USER_COMPONENT)ComponentI;
                      if (GnlUserComponentFormalType (UserCompo) ==
                          GNL_FORMAL_CHAR)
                          GnlUpdateFanoutFromUserCompoCell (UserCompo);
                      else
                         GnlUpdateFanoutFromUserCompo (UserCompo);
                      break;

             case GNL_TRISTATE_COMPO:
                      TristateCompo = (GNL_TRISTATE_COMPONENT)ComponentI;
                      GnlUpdateFanoutFromTriStateCompo (TristateCompo);
                      break;

                 case GNL_BUF_COMPO:
                      BufCompo = (GNL_BUF_COMPONENT)ComponentI;
                      GnlUpdateFanoutFromBufCompo (BufCompo);
                      break;

             case GNL_SEQUENTIAL_COMPO:
                      SeqCompo = (GNL_SEQUENTIAL_COMPONENT)ComponentI;
                      GnlUpdateFanoutFromSeqCompo (SeqCompo);
                      break;

             default:
                      fprintf (stderr, " ERROR: unknow component\n");
                      exit (1);
                }
        }

    for (i=0; i<BListSize (GnlOutputs (Gnl)); i++)
        {
         VarI = (GNL_VAR)BListElt (GnlOutputs (Gnl), i);
         GnlUpdateFanoutOnVarRec (VarI);
        }

    MaxFanout = 0;

    /* We scan all the variables and pick up the max fanout value  */
    HashTableNames = GnlHashNames (Gnl);
    for (i=0; i<BListSize (HashTableNames); i++)
        {
         BucketI = (BLIST)BListElt (HashTableNames, i);
         for (j=0; j<BListSize (BucketI); j++)
             {
              VarJ = (GNL_VAR)BListElt (BucketI, j);
              if ((int)GnlVarHook (VarJ) > MaxFanout)
                 {
                  MaxFanout = (int)GnlVarHook (VarJ);
```

```
                }
            }
        }

    SetGnlMaxFanout (Gnl, MaxFanout);

    /* We reset the Hook field of all the Vars of the Gnl.          */
    GnlResetVarHook (Gnl);
}

/*-------------------------------------------------------------------------*/
/* GnlComputeMaxFanoutInGnlKeepVarHook                                     */
/*-------------------------------------------------------------------------*/
/* This procedure computes the MaxFanout found in the gnl 'Gnl' and        */
/* stores it in the field 'GnlMaxFanout (Gnl)'.                            */
/* We use the field 'GnlVarHook' of each var to store the number of        */
/* times this variable is used.                                            */
/* The GNL must be composed only of components and simple assignements     */
/*-------------------------------------------------------------------------*/
void GnlComputeMaxFanoutInGnlKeepVarHook (Gnl)
    GNL             Gnl;
{
    int                         i;
    GNL_COMPONENT           ComponentI;
    int                         MaxFanout;
    BLIST                   BucketI;
    int                         j;
    GNL_VAR                 VarJ;
    BLIST                   HashTableNames;
    BLIST                   Interface;
    GNL_USER_COMPONENT          UserCompo;
    GNL_TRISTATE_COMPONENT      TristateCompo;
    GNL_BUF_COMPONENT           BufCompo;
    GNL_SEQUENTIAL_COMPONENT    SeqCompo;
    GNL_VAR                 VarI;



    /* We reset the Hook filed of all the Vars of the Gnl.          */
    GnlResetVarHook (Gnl);

    for (i=0; i<BListSize (GnlComponents (Gnl)); i++)
        {
         ComponentI = (GNL_COMPONENT)BListElt (GnlComponents (Gnl), i);
         switch (GnlComponentType (ComponentI)) {
             case GNL_USER_COMPO:
                      UserCompo = (GNL_USER_COMPONENT)ComponentI;
                      GnlUpdateFanoutFromUserCompoKeepVar (UserCompo);
                      break;

             case GNL_TRISTATE_COMPO:
                      TristateCompo = (GNL_TRISTATE_COMPONENT)ComponentI;
                      GnlUpdateFanoutFromTriStateCompo (TristateCompo);
                      break;

             case GNL_SEQUENTIAL_COMPO:
                      SeqCompo = (GNL_SEQUENTIAL_COMPONENT)ComponentI;
```

```
                    GnlUpdateFanoutFromSeqCompo (SeqCompo);
                    break;

               case GNL_BUF_COMPO:
                    BufCompo = (GNL_BUF_COMPONENT)ComponentI;
                    GnlUpdateFanoutFromBufCompo (BufCompo);
                    break;

          default:
                    fprintf (stderr, " ERROR: unknow component\n");
                    exit (1);
               }
        }

    for (i=0; i<BListSize (GnlOutputs (Gnl)); i++)
        {
         VarI = (GNL_VAR)BListElt (GnlOutputs (Gnl), i);
         GnlUpdateFanoutOnVarRec (VarI);
        }

    MaxFanout = 0;

    /* We scan all the variables and pick up the max fanout value  */
    HashTableNames = GnlHashNames (Gnl);
    for (i=0; i<BListSize (HashTableNames); i++)
        {
         BucketI = (BLIST)BListElt (HashTableNames, i);
         for (j=0; j<BListSize (BucketI); j++)
             {
              VarJ = (GNL_VAR)BListElt (BucketI, j);
              if ((int)GnlVarHook (VarJ) > MaxFanout)
                 {
                  MaxFanout = (int)GnlVarHook (VarJ);
                 }
             }
        }

    SetGnlMaxFanout (Gnl, MaxFanout);

}

/*-------------------------------------------------------------------------*/
/* GnlComputeMaxFanoutInNetworkInGnlRec                                    */
/*-------------------------------------------------------------------------*/
void GnlComputeMaxFanoutInNetworkInGnlRec (Nw, Gnl)
   GNL_NETWORK    Nw;
   GNL            Gnl;
{
   int                  i;
   GNL_COMPONENT  ComponentI;
   GNL_USER_COMPONENT   UserCompoI;
   GNL                  GnlCompoI;


   if (GnlTag (Gnl) == GnlNetworkTag (Nw))
      return;
```

```
    SetGnlTag (Gnl, GnlNetworkTag (Nw));

    GnlComputeMaxFanoutInGnlKeepVarHook (Gnl);

    for (i=0; i<BListSize (GnlComponents (Gnl)); i++)
        {
         ComponentI = (GNL_COMPONENT)BListElt (GnlComponents (Gnl), i);
         if (GnlComponentType (ComponentI) != GNL_USER_COMPO)
            continue;

         UserCompoI = (GNL_USER_COMPONENT)ComponentI;
         GnlCompoI = GnlUserComponentGnlDef (UserCompoI);

         if (!GnlCompoI)
            continue;

         GnlComputeMaxFanoutInNetworkInGnlRec (Nw, GnlCompoI);
        }

}


/*-------------------------------------------------------------------------*/
/* GnlComputeMaxFanoutInNetwork                                            */
/*-------------------------------------------------------------------------*/
/* This procedure computes the Fanout of each variable                     */
/*-------------------------------------------------------------------------*/
void GnlComputeMaxFanoutInNetwork (Nw)
   GNL_NETWORK     Nw;
{
   GNL             TopGnl;


   SetGnlNetworkTag (Nw, GnlNetworkTag (Nw)+1);

   TopGnl = GnlNetworkTopGnl (Nw);

   GnlComputeMaxFanoutInNetworkInGnlRec (Nw, TopGnl);

}


/*-------------------------------------------------------------------------*/
/* GnlUpdateComponentGnlDef                                                */
/*-------------------------------------------------------------------------*/
/* This procedure rebuild the list of functions of the gnl 'Gnl' by        */
/* scanning all the variables of the Hash Tables of 'Gnl' and adding       */
/* them if their 'Function' field is not NULL.                             */
/*-------------------------------------------------------------------------*/
GNL_STATUS GnlUpdateGnlFunction (Gnl)
   GNL             Gnl;
{
   BLIST           HashTableNames;
   int                  i;
   BLIST           BucketI;
   int                  j;
   GNL_VAR         VarJ;
```

```
    BSize (GnlFunctions (Gnl)) = 0;

    HashTableNames = GnlHashNames (Gnl);
    for (i=0; i<BListSize (HashTableNames); i++)
        {
         BucketI = (BLIST)BListElt (HashTableNames, i);
         for (j=0; j<BListSize (BucketI); j++)
             {
              VarJ = (GNL_VAR)BListElt (BucketI, j);
              if (GnlVarFunction (VarJ))
                 {
                  if (BListAddElt (GnlFunctions (Gnl), (int)
                                   VarJ))
                     return (GNL_MEMORY_FULL);
                 }
             }
        }

    return (GNL_OK);

}

/*-------------------------------------------------------------------------*/
/* GnlAddOutputsCoupleInverter                                             */
/*-------------------------------------------------------------------------*/
GNL_STATUS GnlAddOutputsCoupleInverter (Gnl)
    GNL             Gnl;
{
    int                 i;
    GNL_VAR         VarI;
    GNL_FUNCTION        FunctionI;
    GNL_NODE        NodeI;
    GNL_NODE        NodeNot;
    GNL_NODE        NodeNotNot;


    for (i=0; i<BListSize (GnlFunctions (Gnl)); i++)
        {
         VarI = (GNL_VAR)BListElt (GnlFunctions (Gnl), i);
         FunctionI = GnlVarFunction (VarI);
         if (!FunctionI)
            continue;
         NodeI = GnlFunctionOnSet (FunctionI);

         if (GnlNodeOp (NodeI) == GNL_CONSTANTE)
            continue;
         if (GnlNodeOp (NodeI) == GNL_VARIABLE)
            continue;

         if (GnlCreateNodeNot (Gnl, NodeI, &NodeNot))
            return (GNL_MEMORY_FULL);
         if (GnlCreateNodeNot (Gnl, NodeNot, &NodeNotNot))
            return (GNL_MEMORY_FULL);

         SetGnlFunctionOnSet (FunctionI, NodeNotNot);
```

```
        }

    return (GNL_OK);
}


/*-------------------------------------------------------------------------*/
/* GnlPrintFormatSignalName                                                */
/*-------------------------------------------------------------------------*/
/* Procedure which prints out a signal name in file 'File' with a          */
/* limited size of 'Length'. The name is then truncated and only the end*/
/* of the name is printed out prefixed by '..'.                            */
/*-------------------------------------------------------------------------*/
void GnlPrintFormatSignalName (File, Name, Length)
    FILE            *File;
    char            *Name;
    int             Length;
{
    int             i;


    if (strlen (Name) >= Length)
        {
         fprintf (File, "..");
         for (i=strlen (Name)-Length+2; i<strlen (Name); i++)
             fprintf (File, "%c", Name[i]);
        }
    else
        {
         for (i=0; i<strlen (Name); i++)
             fprintf (File, "%c", Name[i]);
         while (i<Length)
               {
                fprintf (File, " ");
                i++;
               }
        }

}

/*-------------------------------------------------------------------------*/
/* GnlRemoveFunctionFromGnl                                                */
/*-------------------------------------------------------------------------*/
/* This procedure removes the GNL_VAR 'Func' from the list of functions */
/* of the gnl 'Gnl'.                                                       */
/*-------------------------------------------------------------------------*/
void GnlRemoveFunctionFromGnl (Gnl, Func)
    GNL             Gnl;
    GNL_VAR  Func;
{
    int             i;
    GNL_VAR  FuncI;


    for (i=0; i<BListSize (GnlFunctions (Gnl)); i++)
        {
         FuncI = (GNL_VAR)BListElt (GnlFunctions (Gnl), i);
```

```
        if (FuncI == Func)
           {
            BListDelInsert (GnlFunctions (Gnl), i+1);
            return;
           }
       }
}

/*-------------------------------------------------------------------------*/
/* GnlTrackVarInGnl                                                        */
/*-------------------------------------------------------------------------*/
void GnlTrackVarInGnl (Gnl, VarName)
   GNL             Gnl;
   char            *VarName;
{
   int             i;
   int             j;
   BLIST         BucketI;
   GNL_VAR       VarJ;
   BLIST         HashTableNames;
   GNL_FUNCTION Function;
   GNL_NODE      Node;


for (i=0; i<BListSize (GnlFunctions (Gnl)); i++)
    {
     VarJ = (GNL_VAR)BListElt (GnlFunctions (Gnl), i);
     if (!strcmp (GnlVarName (VarJ), VarName))
        {
         fprintf (stderr, " Oui dans les functions !\n");
         break;
        }
    }

   HashTableNames = GnlHashNames (Gnl);

   fprintf (stderr, "hashtable = %d\n", HashTableNames);
   for (i=0; i<BListSize (HashTableNames); i++)
       {
        BucketI = (BLIST)BListElt (HashTableNames, i);
        for (j=0; j<BListSize (BucketI); j++)
            {
             VarJ = (GNL_VAR)BListElt (BucketI, j);
             if (!strcmp (GnlVarName (VarJ), VarName))
                {
                 fprintf (stderr, "Var = %d\n", VarJ);
                 Function = GnlVarFunction (VarJ);
                 Node = GnlFunctionOnSet (Function);
                 GnlPrintNode (stderr, Node);
                 fprintf (stderr, "\n");
                }
            }
       }

}

/*-------------------------------------------------------------------------*/
```

```
/* PrintListVar                                                                */
/*-----------------------------------------------------------------------------*/
/* Prints out the list of all the GNL_VAR present in the list 'List'           */
/*-----------------------------------------------------------------------------*/
void PrintListVar (List)
   BLIST    List;
{
   int            i;
   GNL_VAR  VarI;

   for (i=0; i<BListSize (List); i++)
       {
        VarI = (GNL_VAR)BListElt (List, i);
        if (!VarI)
           fprintf (stderr, "NULL ");
        fprintf (stderr, "%s ", GnlVarName (VarI));
       }
   fprintf (stderr, "\n");
}

/*-----------------------------------------------------------------------------*/
/* PrintGnl                                                                    */
/*-----------------------------------------------------------------------------*/
void PrintGnl (Gnl)
   GNL      Gnl;
{

   GnlPrintGnl (stderr, Gnl);
}


/*-----------------------------------------------------------------------------*/
/* GnlTrackVarRec                                                              */
/*-----------------------------------------------------------------------------*/
void GnlTrackVarRec (Nw, Gnl, GnlName, VarName)
   GNL_NETWORK    Nw;
   GNL            Gnl;
   char           *GnlName;
   char           *VarName;
{
   int                  i;
   GNL_COMPONENT  ComponentI;
   GNL_USER_COMPONENT   UserCompoI;
   GNL                  GnlCompoI;
   BLIST          Components;


   if (!strcmp (GnlName (Gnl), GnlName))
      GnlTrackVarInGnl (Gnl, VarName);

   Components = GnlComponents (Gnl);
   for (i=0; i<BListSize (Components); i++)
       {
        ComponentI = (GNL_COMPONENT)BListElt (Components, i);
        if (GnlComponentType (ComponentI) != GNL_USER_COMPO)
           continue;
```

```
        UserCompoI = (GNL_USER_COMPONENT)ComponentI;
        GnlCompoI = GnlUserComponentGnlDef (UserCompoI);

        if (!GnlCompoI)
           continue;

        GnlTrackVarRec (Nw, GnlCompoI, GnlName, VarName);
       }
}


/*-------------------------------------------------------------------------*/
/* GnlTrackVar                                                             */
/*-------------------------------------------------------------------------*/
/* Procedure used to debug a specific variable in the network in a         */
/* specific Gnl. You have to specify the Network, the name of the Gnl      */
/* and the name of the variable.                                           */
/* Modify the core of 'GnlTrackVarInGnl' for the fields of the var you     */
/* want to track.                                                          */
/*-------------------------------------------------------------------------*/
void GnlTrackVar (Nw, GnlName, VarName)
   GNL_NETWORK    Nw;
{
   GNL            TopGnl;

   fprintf (stderr, " !!! TRACKING VAR <%s> in [%s]\n",
            VarName, GnlName);
   TopGnl = GnlNetworkTopGnl (Nw);

   GnlTrackVarRec (Nw, TopGnl, GnlName, VarName);
   fprintf (stderr, " !!! TRACKING VAR DONE\n");
}

/*-------------------------------- EOF --------------------------------*/
```

```
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*     File:           libc_api.c                                          */
/*     Version:        1.0                                                 */
/*     Modifications:  -                                                   */
/*     Documentation:  -                                                   */
/*                                                                         */
/*                                                                         */
/*-------------------------------------------------------------------------*/

#include <stdio.h>
#include <malloc.h>
#include "blist.h"
#include "gnl.h"
#include "libc_mem.h"
#include "libc_api.h"



/*-------------------------------------------------------------------------*/
/* EXTERN                                                                  */
/*-------------------------------------------------------------------------*/
extern int              libc_error_count;
extern LIBC_LIB               tech_lib;

/*-------------------------------------------------------------------------*/
/* LibRead                                                                 */
/*-------------------------------------------------------------------------*/
LIB_STATUS LibRead (char *FileName, LIBC_LIB *Library)
{
   int      code;


   code = 0;

   libc_error_count = 0;
   if ((code = parse_a_tlib_file (FileName)) || libc_error_count)
      return (LIB_READ_ERROR);

   *Library = tech_lib;

   return (LIB_OK);
}

/*-------------------------------EOF-------------------------------------*/
```

```
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*     File:           libc_api.e                                          */
/*     Version:        1.0                                                 */
/*     Modifications:  -                                                   */
/*     Documentation:  -                                                   */
/*                                                                         */
/*-------------------------------------------------------------------------*/

extern LIBC_OPER_COND LibGetOperatingConditionsFromName (LIBC_LIB Lib, char
*Name);

extern void LibInitializeDeltaOperCondAndNomOperCond (LIBC_LIB Lib, char
*Name);

extern float LibScalValue (float          Value,
                    float          KProcess,
                    float          KTemp,
                    float          KVoltage);

extern LIBC_PIN LibGetPinFromNameAndCell (LIBC_CELL Cell, char *PinName);

extern LIBC_TIMING LibGetArcTiming (LIBC_PIN PinIn, LIBC_PIN PinOut);

extern float LibGetScaledValFromTransCapaAndMatrix (float Trans, float Capa,
                                    LIBC_TABLE_VAL Matrix,
                                    LIBC_CELL Cell,
                                    LIB_MATRIX_TYPE Type);

extern LIBC_WIRE_LOAD_SELECT LibGetWireLoadSelectFromName (LIBC_LIB Lib, char
*Name);

extern float LibGetWireLengthFromFanout (LIBC_WIRE_LOAD WireL, int Fanout);

extern float LibGetWireLengthFromCapa (LIBC_WIRE_LOAD WireL, float Capa);

extern float LibGetWireScaledCapaFromLength (LIBC_WIRE_LOAD WireL,
                                    float Length, LIBC_LIB Lib);

extern float LibGetWireScaledResiFromLength (LIBC_WIRE_LOAD WireL,
                                    float Length, LIBC_LIB Lib);

extern float LibGetWireAreaFromLength (LIBC_WIRE_LOAD WireL, float Length,
                                LIBC_LIB Lib);

extern void LibDelayArcCellRise (float InTransR, float InTransF, float
OutCapa,
                  LIBC_CELL Cell, char *PinInName, char *PinOutName,
                   float *DelayR, float *OutTransR);

extern void LibDelayArcCellFall (float InTransR, float InTransF, float
OutCapa,
                   LIBC_CELL Cell, char *PinInName, char *PinOutName,
                   float *DelayF, float *OutTransF);

extern void LibDelayArcCell (float InTransR, float InTransF, float OutCapa,
                  float WireResistance, int Fanout,
```

```
                  LIBC_CELL Cell, LIBC_PIN PinInName, LIBC_PIN PinOutName,
                  float *DelayR, float *OutTransR,
                  float *DelayF, float *OutTransF);

extern void LibDelayArcCellRiseSetup (float InTransR, float InTransF,
                  float OutCapa, LIBC_CELL Cell, LIBC_PIN PinIn,
                  LIBC_PIN PinClock, float *DelayR);

extern void LibDelayArcCellFallSetup (float InTransR, float InTransF,
                  float OutCapa, LIBC_CELL Cell, LIBC_PIN PinIn,
                  LIBC_PIN PinClock,float *DelayF);
```

```
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*     File:           libc_api.h                                          */
/*     Version:        1.0                                                 */
/*     Modifications:  -                                                   */
/*     Documentation:  -                                                   */
/*                                                                         */
/*-------------------------------------------------------------------------*/


/*-------------------------------------------------------------------------*/
/* LIB_STATUS                                                              */
/*-------------------------------------------------------------------------*/
typedef enum {
      LIB_OK,
      LIB_READ_ERROR,
      LIB_MEMORY_FULL
} LIB_STATUS;


/*-------------------------------------------------------------------------*/
/* LIB_STATUS                                                              */
/*-------------------------------------------------------------------------*/
typedef enum {
      MATRIX_CELL_RISE,
      MATRIX_CELL_FALL,
      MATRIX_RISE_PROPAGATION,
      MATRIX_FALL_PROPAGATION,
      MATRIX_RISE_TRANSITION,
      MATRIX_FALL_TRANSITION,
      MATRIX_SETUP_RISE,
      MATRIX_SETUP_FALL
} LIB_MATRIX_TYPE;


/*-------------------------------------------------------------------------*/
/* API type definitions                                                    */
/*-------------------------------------------------------------------------*/
/* Defintion of pointer-based types related to the original types          */
/* defined in file "libcds.h".                                             */
/*-------------------------------------------------------------------------*/
typedef struct libc_lib_rec         *LIBC_LIB;
typedef struct libc_cell_rec        *LIBC_CELL;
typedef struct libc_pin_rec         *LIBC_PIN;
typedef struct libc_bool_opr_rec    *LIBC_BOOL_OPR;
typedef struct libc_name_list_rec   *LIBC_NAME_LIST;
typedef struct libc_k_factor_rec    *LIBC_KFATOR;
typedef struct libc_operating_condition_rec     *LIBC_OPER_COND;
typedef struct libc_oc_power_rail_rec     *LIBC_OC_POWER_RAIL;
typedef struct libc_fanout_length_rec     *LIBC_FAN_LEN;
typedef struct libc_wire_load_rec   *LIBC_WIRE_LOAD;
typedef struct libc_wire_load_selection_rec     *LIBC_WIRE_LOAD_SELECT;
typedef struct libc_wire_load_from_area_rec     *LIBC_WIRE_LOAD_FROM_AREA;
typedef struct libc_ff_latch_rec    *LIBC_FF_LATCH;
typedef struct libc_table_val_rec   *LIBC_TABLE_VAL;
typedef struct libc_timing_rec            *LIBC_TIMING;
```

```
/*---------------------------------------------------------------------*/
/* LIBC_NAME_LIST                                            */
/*---------------------------------------------------------------------*/
#define LibNameListName(l)          ((l)->name)
#define SetLibNameListName(l,n)          ((l)->name = n)

#define LibNameListNext(l)          ((l)->next)
#define SetLibNameListNext(l,n)          ((l)->next = n)

/*---------------------------------------------------------------------*/
/* LIBC_BOOL_OPR                                             */
/*---------------------------------------------------------------------*/
/* Selectors related to the LIBC_BOOL_OPR group.                       */
/*---------------------------------------------------------------------*/
#define LibBoolOprType(m)           ((m)->type)
#define SetLibBoolOprType(m,c)           ((m)->type = c)

#define LibBoolOprValue(m)          ((m)->u.value)
#define SetLibBoolOprValue(m,c)          ((m)->u.value = c)

#define LibBoolOprIdName(m)         ((text_buffer*)(m)->u.id_name)
#define SetLibBoolOprIdName(m,c)    ((m)->u.id_name = (text_buffer*) c)

#define LibBoolOprLeftSon(m)        ((m)->L)
#define SetLibBoolOprLeftSon(m,c)   ((m)->L = c)

#define LibBoolOprRightSon(m)       ((m)->R)
#define SetLibBoolOprRightSon(m,c)  ((m)->R = c)

/*---------------------------------------------------------------------*/
/* LIBC_LIB                                                  */
/*---------------------------------------------------------------------*/
/* Selectors related to the LIB group.                                 */
/*---------------------------------------------------------------------*/
#define LibName(m)                  ((text_buffer*)(m)->lib_name)
#define SetLibName(m,c)             ((m)->lib_name = (text_buffer*)c)

#define LibCells(m)                 ((m)->cells)
#define SetLibCells(m,c)            ((m)->cells = c)

#define LibHook(m)                  ((m)->hook)
#define SetLibHook(m,c)             ((m)->hook = c)

#define LibDelayModel(Lib)          ((Lib)->delay_model)
#define LibTimeUnit(Lib)            ((Lib)->time_unit)
#define LibNomProcess(Lib)          ((Lib)->nom_process)
#define LibNomTemp(Lib)             ((Lib)->nom_temperature)
#define LibNomVolt(Lib)             ((Lib)->nom_voltage)

/* ---- lib_group_stmt */

#define LibLutTemplate(Lib)         ((Lib)->lut_template)
#define LibPlutTemplate(Lib)        ((Lib)->plut_template)
#define LibOperatingCond(Lib)       ((Lib)->operating_cond)
#define LibWireLoad(Lib)            ((Lib)->wire_load)
#define LibWireLoadSelect(Lib)           ((Lib)->wire_load_selection)
```

```
/* ---- default attr */

#define LibDefaultCellLeakagePower(Lib)    ((Lib)->default_cell_leakage_power)
#define LibDefaultCellPower(Lib)     ((Lib)->default_cell_power)

#define LibDefaultFallWireResistance(Lib) \
      ((Lib)->default_fall_wire_resistance)
#define LibDefaultWireLoad(Lib)            ((Lib)->default_wire_load)
#define LibDefaultWireLoadArea(Lib)  ((Lib)->default_wire_load_area)
#define DefaultWireLoadCapa(Lib)     ((Lib)->default_wire_load_capacitance)
#define LibDefaultWireLoadResistance(Lib) \
      ((Lib)->default_wire_load_resistance)
#define LibDefaultWireLoadSelect(Lib)      ((Lib)-
>default_wire_load_selection)
#define LibDefaultFanoutLoad(Lib)    ((Lib)->default_fanout_load)
#define LibDefaultInoutPinCap(Lib)   ((Lib)->default_inout_pin_cap)
#define LibDefaultInputPinCap(Lib)   ((Lib)->default_input_pin_cap)
#define LibDefaultMaxCapa(Lib)       ((Lib)->default_max_capacitance)
#define LibDefaultMaxFanout(Lib)     ((Lib)->default_max_fanout)
#define LibDefaultMaxTransition(Lib)       ((Lib)->default_max_transition)
#define LibDefaultOperatingCond(Lib)       ((Lib)-
>default_operating_conditions)
#define LibDefaultOutputPinCap(Lib)  ((Lib)->default_output_pin_cap)


#define LibKFactor(Lib)              ((Lib)->k_factor)
#define LibScKFactor(Lib)            ((Lib)->sc_k_factor)


/*-------------------------------------------------------------------------*/
/* LIBC_CELL                                                               */
/*-------------------------------------------------------------------------*/
/* Selectors related to the CELL group.                                    */
/*-------------------------------------------------------------------------*/
#define LibCellArea(c)               ((c)->area)
#define SetLibCellArea(c,n)          ((c)->area = n)

#define LibCellName(c)               ((c)->cell_name)
#define SetLibCellName(c,n)          ((c)->cell_name = n)

#define LibCellPins(c)               ((c)->pins)
#define SetLibCellPins(c,n)          ((c)->pins = n)

#define LibCellNext(c)               ((c)->next)
#define SetLibCellNext(c,n)          ((c)->next = n)

#define LibCellHook(l)               ((l)->hook)
#define SetLibCellHook(l,c)          ((l)->hook = c)

#define LibCellPower(Cell)                ((Cell)->cell_power)
#define LibCellCellLeakagePower(Cell)          ((Cell)->cell_leakage_power)
#define LibCellDontTouch(Cell)                 ((Cell)->dont_touch)
#define LibCellDontUse(Cell)              ((Cell)->dont_use)
#define LibCellPad(Cell)                  ((Cell)->pad_cell)
#define LibCellPadType(Cell)              ((Cell)->pad_type)
#define LibCellScalingFactors(Cell)       ((Cell)->_scaling_factors)
#define LibCellScanGroup(Cell)                 ((Cell)->scan_group)
```

```
#define LibCellPinEqual(Cell)              ((Cell)->pin_equal)
#define LibCellPinOpposite(Cell)           ((Cell)->pin_opposite)

#define LibCellInternalPowerC(Cell)        ((Cell)->internal_power_c)
#define LibCellLeakagePower(Cell)          ((Cell)->leakage_power)

#define LibCellFFLatch(Cell)               ((Cell)->ff_latch)

/*--------------------------------------------------------------------*/
/* LIBC_PIN                                                           */
/*--------------------------------------------------------------------*/
/* Selectors related to the PIN group.                                */
/*--------------------------------------------------------------------*/
#define LibPinName(p)                ((p)->pin_name)
#define LibPinNameFirst(p)           (((p)->pin_name)->name)
#define SetLibPinName(p,d)           ((p)->pin_name = d)

#define LibPinDirection(p)           ((p)->direction)
#define SetLibDirection(p,d)         ((p)->direction = d)

#define LibPinFunction(p)            ((p)->function)
#define SetLibPinFunction(p,f)             ((p)->function = f)

#define LibPinNext(p)                ((p)->next)
#define SetLibPinNext(p,n)           ((p)->next = n)

#define LibPinHook(l)                ((l)->hook)
#define SetLibPinHook(l,c)           ((l)->hook = c)

#define LibPinCapa(Pin)                    ((Pin)->capacitance)
#define LibPinClock(Pin)                   ((Pin)->clock)
#define LibPinClockGateEnable_pin(Pin)           ((Pin)-
>clock_gate_enable_pin)
#define LibPinFanoutLoad(Pin)              ((Pin)->fanout_load)

#define LibPinInvertedOutput(Pin)          ((Pin)->inverted_output)
#define LibPinIsPad(Pin)                   ((Pin)->is_pad)
#define LibPinMaxFanout(Pin)               ((Pin)->max_fanout)
#define LibPinMaxTransition(Pin)           ((Pin)->max_transition)
#define LibPinMaxCapa(Pin)                 ((Pin)->max_capacitance)
#define LibPinMinFanout(Pin)               ((Pin)->min_fanout)
#define LibPinMinTransition(Pin)           ((Pin)->min_transition)
#define LibPinMinCapa(Pin)                 ((Pin)->min_capacitance)
#define LibPinNextstateType(Pin)           ((Pin)->nextstate_type)
#define LibPin3State(Pin)                  ((Pin)->three_state)
#define LibPinStateFunction(Pin)           ((Pin)->state_function)

#define LibPinTiming(Pin)                  ((Pin)->timing)
#define LibPinMinPulseWidth(Pin)           ((Pin)->min_pulse_width)
#define LibPinMinimumPeriod(Pin)           ((Pin)->minimum_period)
#define LibPinInternalPower(Pin)           ((Pin)->internal_power)
#define LibPinType(Pin)                    ((Pin)->pin_type)

/*--------------------------------------------------------------------*/
/* LIBC_KFATOR                                                        */
/*--------------------------------------------------------------------*/
/* Selectors related to the KFATOR group.                             */
```

```
/*-------------------------------------------------------------------------*/
#define LibKFactorCellRise(KF)                          ((KF)->k_cell_rise)
#define LibKFactorCellFall(KF)                          ((KF)->k_cell_fall)
#define LibKFactorCellLeakagePower(KF)                  ((KF)->k_cell_leakage_power)
#define LibKFactorCellPower(KF)                         ((KF)->k_cell_power)
#define LibKFactorFallPropagation(KF)                   ((KF)->k_fall_propagation)
#define LibKFactorFallTransition(KF)                    ((KF)->k_fall_transition)
#define LibKFactorRisePropagation(KF)                   ((KF)->k_rise_propagation)
#define LibKFactorRiseTransition(KF)                    ((KF)->k_rise_transition)
#define LibKFactorFallWireResistance(KF)   ((KF)->k_fall_wire_resistance)
#define LibKFactorHoldFall(KF)                          ((KF)->k_hold_fall)
#define LibKFactorHoldRise(KF)                          ((KF)->k_hold_rise)
#define LibKFactorPinCap(KF)               ((KF)->k_pin_cap)
#define LibKFactorPinPower(KF)                          ((KF)->k_pin_power)
#define LibKFactorRiseWireResistance(KF)   ((KF)->k_rise_wire_resistance)
#define LibKFactorSetupFall(KF)                         ((KF)->k_setup_fall)
#define LibKFactorSetupRise(KF)                         ((KF)->k_setup_rise)
#define LibKFactorSkewFall(KF)                          ((KF)->k_skew_fall)
#define LibKFactorSkewRise(KF)                          ((KF)->k_skew_rise)
#define LibKFactorSlopeFall(KF)                         ((KF)->k_slope_fall)
#define LibKFactorSlopeRise(KF)                         ((KF)->k_slope_rise)
#define LibKFactorWireCap(KF)              ((KF)->k_wire_cap)
#define LibKFactorWireRes(KF)              ((KF)->k_wire_res)

/*-------------------------------------------------------------------------*/
/* LIBC_OPER_COND                                                          */
/*-------------------------------------------------------------------------*/
/* Selectors related to the LIBC_OPER_COND group.                          */
/*-------------------------------------------------------------------------*/

#define LibOperCondOcName(OperCond)        ((OperCond)->oc_name)
#define LibOperCondProcess(OperCond)             ((OperCond)->process)
#define LibOperCondTemp(OperCond)          ((OperCond)->temperature)
#define LibOperCondVolt(OperCond)          ((OperCond)->voltage)
#define LibOperCondTreeType(OperCond)            ((OperCond)->tree_type)
#define LibOperCondPowerRail(OperCond)           ((OperCond)->power_rail)
#define LibOperCondNext(OperCond)          ((OperCond)->next)

/*-------------------------------------------------------------------------*/
/* LIBC_OC_POWER_RAIL                                                      */
/*-------------------------------------------------------------------------*/
/* Selectors related to the OC_POWER_RAIL group.                           */
/*-------------------------------------------------------------------------*/

#define LibOCPowerRailPowerSupply(OCPowerRail)  ((OCPowerRail)->power_supply)
#define LibOCPowerRailVolt(OCPowerRail)         ((OCPowerRail)->volt)
#define LibOCPowerRailNext(OCPowerRail)         ((OCPowerRail)->next)

/*-------------------------------------------------------------------------*/
/* LIBC_FAN_LEN                                                            */
/*-------------------------------------------------------------------------*/
/* Selectors related to the FAN_LEN.                                       */
/*-------------------------------------------------------------------------*/

#define LibFanLenFanout(FanLength)         ((FanLength)->fanout)
#define LibFanLenLength(FanLength)         ((FanLength)->length)
#define LibFanLenCapa(FanLength)           ((FanLength)->capacitance)
```

```
#define LibFanLenResistance(FanLength)          ((FanLength)->resistance)
#define LibFanLenArea(FanLength)          ((FanLength)->area)
#define LibFanLenNext(FanLength)          ((FanLength)->next)

/*----------------------------------------------------------------------*/
/* LIBC_WIRE_LOAD                                                       */
/*----------------------------------------------------------------------*/
/* Selectors related to the WIRE_LOAD group.                            */
/*----------------------------------------------------------------------*/

#define LibWireLoadName(WireLoad)          ((WireLoad)->wl_name)
#define LibWireLoadArea(WireLoad)          ((WireLoad)->area)
#define LibWireLoadCapa(WireLoad)          ((WireLoad)->capacitance)
#define LibWireLoadResistance(WireLoad)          ((WireLoad)->resistance)
#define LibWireLoadSlope(WireLoad)          ((WireLoad)->slope)
#define LibWireLoadFanLen(WireLoad)          ((WireLoad)->fanout_length)
#define LibWireLoadNext(WireLoad)          ((WireLoad)->next)

/*----------------------------------------------------------------------*/
/* LIBC_WIRE_LOAD_FROM_AREA                                             */
/*----------------------------------------------------------------------*/
/* Selectors related to the WIRE_LOAD_FROM_AREA.                        */
/*----------------------------------------------------------------------*/

#define LibWireLFromAreaMinArea(WireLFromArea)   ((WireLFromArea)->min_area)
#define LibWireLFromAreaMaxArea(WireLFromArea)   ((WireLFromArea)->max_area)
#define LibWireLFromAreaLModel(WireLFromArea)    ((WireLFromArea)-
>_load_model)
#define LibWireLFromAreaNext(WireLFromArea)      ((WireLFromArea)->next)

/*----------------------------------------------------------------------*/
/* LIBC_WIRE_LOAD_SELECT                                                */
/*----------------------------------------------------------------------*/
/* Selectors related to the WIRE_LOAD_SELECT group.                     */
/*----------------------------------------------------------------------*/

#define LibWireLoadSelName(WireLoadSel)          ((WireLoadSel)->wls_name)
#define LibWireLoadSelTable(WireLoadSel)   ((WireLoadSel)->area_table)
#define LibWireLoadSelNext(WireLoadSel)          ((WireLoadSel)->next)

/*----------------------------------------------------------------------*/
/* LIBC_FF_LATCH                                                        */
/*----------------------------------------------------------------------*/
/* Selectors related to the FF_LATCH group.                             */
/*----------------------------------------------------------------------*/

#define LibFFLatchQName(f)          ((f)->Q_name)
#define SetLibFFLatchQName(f,n)          ((f)->Q_name = n)

#define LibFFLatchQNName(f)          ((f)->QN_name)
#define SetLibFFLatchQNName(f,n)     ((f)->QN_name = n)

/* 1: normal, >1 bank                                                   */
#define LibFFLatchWidth(f)          ((f)->width)
#define SetLibFFLatchWidth(f,n)          ((f)->width = n)

#define LibFFLatchIsFF(f)          ((f)->is_ff)
```

```
#define SetLibFFLatchIsFF(f,n)            ((f)->is_ff = n)

/* old model                                                    */
#define LibFFLatchIsState(f)        ((f)->is_state)
#define SetLibFFLatchIsState(f,n)   ((f)->is_state = n)

/* L, H, N, T, X                                               */
#define LibFFLatchClPrVar1(f)       ((f)->clear_preset_var1)
#define SetLibFFLatchClPrVar1(f,n)  ((f)->clear_preset_var1 = n)

/* L, H, N, T, X                                               */
#define LibFFLatchClPrVar2(f)       ((f)->clear_preset_var2)
#define SetLibFFLatchClPrVar2(f,n)  ((f)->clear_preset_var2 = n)

#define LibFFLatchClear(f)          ((f)->clear)
#define SetLibFFLatchClear(f,n)           ((f)->clear = n)

#define LibFFLatchPreset(f)         ((f)->preset)
#define SetLibFFLatchPreset(f,n)    ((f)->preset = n)

/* ff group                                                    */
#define LibFFLatchClockOn(f)        ((f)->clock_on)
#define SetLibFFLatchClockOn(f,n)   ((f)->clock_on = n)

/* ff group                                                    */
#define LibFFLatchNextState(f)            ((f)->next_state)
#define SetLibFFLatchNextState(f,n) ((f)->next_state = n)

/* clock_on_also, enable_on_also                               */
#define LibFFLatchOnAlso(f)         ((f)->on_also)
#define SetLibFFLatchOnAlso(f,n)    ((f)->on_also = n)

/* latch group                                                 */
#define LibFFLatchEnable(f)         ((f)->enable)
#define SetLibFFLatchEnable(f,n)    ((f)->enable = n)

/* latch group                                                 */
#define LibFFLatchDataIn(f)         ((f)->data_in)
#define SetLibFFLatchDataIn(f,n)    ((f)->data_in = n)

/* state group                                                 */
#define LibFFLatchForce00(f)        ((f)->force_00)
#define SetLibFFLatchForce00(f,n)   ((f)->force_00 = n)

/* state group                                                 */
#define LibFFLatchForce01(f)        ((f)->force_01)
#define SetLibFFLatchForce01(f,n)   ((f)->force_01 = n)

/* state group                                                 */
#define LibFFLatchForce10(f)        ((f)->force_10)
#define SetLibFFLatchForce10(f,n)   ((f)->force_10 = n)

/* state group                                                 */
#define LibFFLatchForce11(f)        ((f)->force_11)
#define SetLibFFLatchForce11(f,n)   ((f)->force_11 = n)
```

```
/*-------------------------------------------------------------------------*/
/* LIBC_TABLE_VAL                                                          */
/*-------------------------------------------------------------------------*/
/* Selectors related to the TABLE_VAL.                                     */
/*-------------------------------------------------------------------------*/

#define LibTableValTbl(Table)              ((Table)->_tbl)
#define LibTableValIndex1(Table)           ((Table)->index1)
#define LibTableValIndex2(Table)           ((Table)->index2)
#define LibTableValIndex3(Table)           ((Table)->index3)
#define LibTableValScalar_val(Table)            ((Table)->scalar_val)
#define LibTableValValues(Table)           ((Table)->values)

/*-------------------------------------------------------------------------*/
/* LIBC_TIMING                                                             */
/*-------------------------------------------------------------------------*/
/* Selectors related to the TIMING group.                                  */
/*-------------------------------------------------------------------------*/

#define LibTimingRelatedPin(Timing)        ((Timing)->related_pin)
#define LibTimingType(Timing)              ((Timing)->timing_type)
#define LibTimingSense(Timing)                  ((Timing)->timing_sense)

#define LibTimingCellRise(Timing)          ((Timing)->cell_rise)
#define LibTimingCellFall(Timing)          ((Timing)->cell_fall)
#define LibTimingRisePropagation(Timing)   ((Timing)->rise_propagation)
#define LibTimingFallPropagation(Timing)   ((Timing)->fall_propagation)
#define LibTimingRiseTransition(Timing)         ((Timing)->rise_transition)
#define LibTimingFallTransition(Timing)         ((Timing)->fall_transition)
#define LibTimingRiseConstraint(Timing)         ((Timing)->rise_constraint)
#define LibTimingFallConstraint(Timing)         ((Timing)->fall_constraint)

#define LibTimingNext(Table)               ((Timing)->next)

/*-------------------------------------------------------------------------*/

/*------------ Global Variable of Timing Analyser -----------------*/

/* we have difined this global variable because
we cosult it always durring the timing analyser */

extern float G_DeltaProcess;
extern float G_DeltaTemp;
extern float G_DeltaVoltage;
extern LIBC_OPER_COND G_OperCond;
extern LIBC_WIRE_LOAD G_WireLoad; /* The wire load corresponding a given
                                     netlist. This wire_load is choosed in
                                     the library taking into account the
                                     global area of the netlist.*/

/*------------------------------- EOF -------------------------------*/
```

```
/*================================================

   libc_cell.c -- routines handle cell define


 ================================================ */

#include "libc_def.h"

/* ================================================ */

public
void libc_cell_init(
      text_buffer *cell_name)
{
  lib_cell = new_libc_cell_rec();
  lib_cell->_tlib     = tech_lib;
  lib_cell->cell_name = cell_name;

  /* ---- set default value */
  lib_cell->cell_leakage_power      = tech_lib->default_cell_leakage_power;
  lib_cell->cell_power          = tech_lib->default_cell_power;
}

/* ================================================ */

public
void libc_cell_pin_init(
      libc_name_list_rec *pin_name)
{ libc_timing_rec *tp;

  if (lib_pin != NULL && lib_pin->members != NULL) {
    /* ---- pin group inside the bundle group */
    lib_bundle = lib_pin;           /* save */

    /* ---- allocate memory and set default value */
    lib_pin = copy_libc_pin_rec(lib_bundle);
    free_libc_name_list_rec(lib_pin->members);
    lib_pin->members = NULL;
    for(tp=lib_pin->timing;tp!=NULL;tp=tp->next)
      tp->_current_pin = lib_pin;
    lib_pin->pin_name           = pin_name;
  }
  else if (lib_pin != NULL && lib_pin->is_bus) {
    /* ---- pin group inside the bus group
     *      keep the same lib_pin          */
    lib_bus = lib_pin;        /* save */
    /* ---- compare the pin name must cover all bus range */

    /* ==== NOT IMPLEMENT YET ==== */

    free_libc_name_list_rec(pin_name);
  }
  else {
    lib_pin = new_libc_pin_rec();
```

```
    /* ---- set default value */
    lib_pin->fanout_load      = tech_lib->default_fanout_load;

    lib_pin->max_fanout       = tech_lib->default_max_fanout;
    lib_pin->max_transition   = tech_lib->default_max_transition;
    lib_pin->max_capacitance  = tech_lib->default_max_capacitance;

    /* ---- for ECL model */
    lib_pin->pin_power        = tech_lib->default_pin_power;

    lib_pin->fall_wor_emitter = tech_lib->default_fall_wor_emitter;
    lib_pin->fall_wor_intercept     = tech_lib->default_fall_wor_intercept;
    lib_pin->rise_wor_emitter = tech_lib->default_rise_wor_emitter;
    lib_pin->rise_wor_intercept     = tech_lib->default_rise_wor_intercept;

    lib_pin->emitter_count    = tech_lib->default_emitter_count;
    lib_pin->pin_name           = pin_name;
  }
  lib_pin->_current_cell  = lib_cell;
}

/* ------------------------------------------------ */

public
void libc_cell_pin_default(void)
{ switch (lib_pin->direction) {
    case INPUT_E :
      if (lib_pin->capacitance == 0.0)
        lib_pin->capacitance = tech_lib->default_input_pin_cap;
      break;
    case OUTPUT_E :
      if (lib_pin->capacitance == 0.0)
        lib_pin->capacitance = tech_lib->default_output_pin_cap;
      break;
    case INOUT_E :
      if (lib_pin->capacitance == 0.0)
        lib_pin->capacitance = tech_lib->default_inout_pin_cap;
      break;
  }
}

/* ================================================ */

public
libc_pin_rec *libc_cell_find_pin_by_name(
      libc_cell_rec *cell,
      char *pin_name,
      int index)        /* < -10000 : no index */
{ libc_pin_rec *pp;
  libc_name_list_rec *np;
  char id_name_idx[128],id_name[128],bus_name[128];

  sprintf(id_name_idx,tech_lib->bus_naming_style,pin_name,index);
  sprintf(id_name,"%s",pin_name);

  for (pp=cell->pins;pp!=NULL;pp=pp->next) {
    if (pp->is_bus) {
```

```
        int i,from,to,temp;

        /* ---- whole bus pin */
        for (np=pp->pin_name;np!=NULL;np=np->next) {
          if (strcmp(np->name,id_name) == 0)
            return(pp);                /* FOUND */
        }
        /* ---- indexed bus pin */
        from = pp->_bus_type->bit_from;
        to   = pp->_bus_type->bit_to;
        if (from > to) {
          temp = from; from = to; to = temp;
        }
        for (i=from;i<=to;i++) {
        sprintf(bus_name,tech_lib->bus_naming_style,pp->pin_name->name,i);
          /* if (strcmp(bus_name,id_name) == 0)
           *  return(pp);                * FOUND */
          if (strcmp(bus_name,id_name_idx) == 0)
            return(pp);                /* FOUND */
        }
      }
      else {        /* ---- bundle and normal pin */
        for (np=pp->pin_name;np!=NULL;np=np->next) {
          if (strcmp(np->name,id_name) == 0)
            return(pp);                /* FOUND */
        }
      }
    }
    return(NULL);
}

/* ------------------------------------------------ */

public
int libc_cell_get_bundle_idx(
      libc_cell_rec *cell,
      char *pin_name)
{ libc_pin_rec *pp;
  libc_name_list_rec *np;
  int i;

  for (pp=cell->pins;pp!=NULL;pp=pp->next) {
    if (pp->members == NULL)
      continue;
    for (np=pp->members,i=0;np!=NULL;np=np->next,i++) {
      if (strcmp(np->name,pin_name) == 0)
        return(i);
    }
  }
  return(-99999);
}

/* ------------------------------------------------ */

static
libc_pin_rec *libc_cell_find_pin_rec(
      libc_cell_rec *cell,
```

```
      char *pin_name,
      int index)        /* < -10000 : no index */
{ libc_pin_rec *pp;
  libc_name_list_rec *np;
  char id_name[128];

  if (index >= -10000)
    sprintf(id_name,tech_lib->bus_naming_style,pin_name,index);
  else
    sprintf(id_name,"%s",pin_name);

  for (pp=cell->pins;pp!=NULL;pp=pp->next) {
    if (pp->members || pp->is_bus)        /* no bundle and bus group */
      continue;
    for (np=pp->pin_name;np!=NULL;np=np->next) {
      if (strcmp(np->name,id_name) == 0)
        return(pp);                  /* FOUND */
    }
  }
  return(NULL);
}

/* ------------------------------------------------ */

public
void libc_cell_bundle_post(
      libc_pin_rec *bundle)
{ libc_cell_rec *cell;
  libc_name_list_rec *mp,*np,*np_head=NULL;
  libc_pin_rec *pp;
  libc_timing_rec *tp;

  cell = bundle->_current_cell;

  /* ---- duplicate the members pin */
  for (mp=bundle->members;mp!=NULL;mp=mp->next) {
    pp = libc_cell_find_pin_rec(cell,mp->name,-99999);
    if (pp == NULL) {          /* no entry, need to duplicate */
      np       = new_libc_name_list_rec();
      np->name = copy_string(mp->name);
      np->next = np_head;
      np_head  = np;
    }
  }

  if (np_head != NULL) {
    pp = copy_libc_pin_rec(bundle);
    free_libc_name_list_rec(pp->members);
    pp->members = NULL;
    free_libc_name_list_rec(pp->pin_name);
    pp->pin_name = np_head;
    for(tp=pp->timing;tp!=NULL;tp=tp->next)
      tp->_current_pin = pp;
    pp->next     = cell->pins;
    cell->pins   = pp;
  }
}
```

```
/* ================================================ */

public
libc_name_list_rec *libc_cell_bus_name(
      libc_pin_rec *bus)
{ libc_name_list_rec *np,*np_head = NULL;
  int from,to,i;
  char name[128];

  assert(bus->is_bus);
  from = bus->_bus_type->bit_from;
  to   = bus->_bus_type->bit_to;
  if (from > to) {
    i = from; from = to; to = i;
  }
  for (i=to;i>=from;i--) {
    np = new_libc_name_list_rec();
    sprintf(name,tech_lib->bus_naming_style,bus->pin_name->name,i);
    np->name = copy_string(name);
    np->next = np_head;
    np_head  = np;
  }
  return(np_head);
}

/* ------------------------------------------------ */

/* ---- return 1 : in name is slice buse name : BUS[0:3]
 *      return 0 : A, D[0], ...
 */

static
int libc_cell_ext_bus_name(
      char *name,
      libc_name_list_rec **head,
      libc_name_list_rec **tail)
{ int from_bit,to_bit,tmp;
  char *s,*val,*t;
  libc_name_list_rec *np;
  char bus_n[256];

  (*head) = (*tail) = NULL;
  for (s=name;(*s)!='\0'&&(*s)!='[';s++);
  if ((*s) == '\0')
    return(0);
  t = s;
  (*s) = '\0';
  val = ++s;
  for (;(*s)!='\0'&&(*s)!=']'&&(*s)!=':';s++);
  if ((*s) == ']' || (*s) == '\0') {
    (*t) = '[';
    return(0);
  }
  (*s) = '\0';
  from_bit = atoi(val);
  (*s) = ':';
```

```
  val = ++s;
  for (;(*s)!='\0'&&(*s)!=']';s++);
  (*s) = '\0';
  to_bit = atoi(val);
  (*s) = ']';

#if 0
  if (from_bit > to_bit) {
    tmp = from_bit; from_bit = to_bit; to_bit = tmp;
  }

  for (tmp=from_bit;tmp<=to_bit;tmp++) {
    sprintf(bus_n,tech_lib->bus_naming_style,name,tmp);
    np       = new_libc_name_list_rec();
    np->name = copy_string(bus_n);
    if ((*head) == NULL)
      (*head) = np;
    else
      (*tail)->next = np;
    (*tail) = np;
  }
#else
  /* ---- may be bug here */
  np       = new_libc_name_list_rec();
  np->name = copy_string(name);
  (*head) = (*tail) = np;
#endif
  (*t) = '[';
  return(1);
}

/* ------------------------------------------------ */

public
void libc_cell_relative_pin_handle(
      libc_name_list_rec *head)
{ libc_name_list_rec *np,*next_np,*nhp,*ntp;

  for (np=head;np!=NULL;np=next_np) {
    next_np = np->next;
    if (libc_cell_ext_bus_name(np->name,&nhp,&ntp)) {
      free_text_buffer(np->name);
      np->name = nhp->name;
      nhp->name = NULL;
      if (nhp != ntp) {
        ntp->next = np->next;
        np->next  = nhp->next;
        nhp->next = NULL;
      }
      free_libc_name_list_rec(nhp);
    }
  }
}

/* ================================================ */
/* ================================================ */
```

```
public
void libc_cell_name_list_list(
      libc_name_list_list **head,
      libc_name_list_rec *name_list1,
      libc_name_list_rec *name_list2)
{ libc_name_list_list *nll;

  nll = new_libc_name_list_list();
  nll->name_list1 = name_list1;
  nll->name_list2 = name_list2;
  nll->next = (*head);
  (*head) = nll;
}

/* ================================================= */

public
void libc_cell_find_pin_type(
      text_buffer *type_name)
{ libc_type_rec *p;

  for (p=tech_lib->type;p!=NULL;p=p->next) {
    if (strcmp(p->type_name,type_name) == 0) {
      lib_pin->_bus_type = p;
      break;
    }
  }
  free_text_buffer(type_name);
}

/* ================================================= */

public
libc_k_factor_rec *libc_cell_k_factor(void)
{
  if (sc_kfc != NULL)
    return(sc_kfc);
  return(tech_lib->k_factor);
}

/* ------------------------------------------------- */

public
libc_k_factor_rec *libc_cell_find_sc_group(
      text_buffer *name)
{ libc_k_factor_rec *p;
  char msg[128];

  for (p=tech_lib->sc_k_factor;p!=NULL;p=p->next) {
    if (strcmp(name,p->kf_name) == 0) {
      free_text_buffer(name);
      return(p);
    }
  }
  sprintf(msg,"scaling_factors group (%s) not found.",name);
  libcerror(msg);
  free_text_buffer(name);
```

```
  return(NULL);
}

/* ================================================= */

/* ---- ff,latch group */

public
void libc_cell_preset_var(
      char *preset_var,
      text_buffer *str)
{
  (*preset_var) = 'X';
  if (str[0] == '\0' || str[1] != '\0')
    libcerror("only L,H,N,T,X is allowed.");
  else if (str[0] != 'L' && str[0] != 'H' && str[0] != 'N' &&
           str[0] != 'T' && str[0] != 'X')
    libcerror("only L,H,N,T,X is allowed.");
  else
    (*preset_var) = str[0];
  free_text_buffer(str);
}

/* ================================================= */
/* for test_cell */

/* ------------------------------------------------- */

public
void libc_cell_pin_type_assign(
      libc_name_list_rec *pin_names,
      signal_type_E sig_type)
{ libc_pin_rec *pin;
  libc_name_list_rec *np;

  if (pin_names == NULL)
    return;

  for (np=pin_names;np!=NULL;np=np->next) {
    pin = libc_cell_find_pin_by_name(lib_tc_cell,np->name,-99999);
    if (pin == NULL)
      continue;

    switch(sig_type) {
      case ST_TSI_E :
      case ST_TSII_E :
        pin->pin_type |= l_axubPortTypeScanIn;
        break;
      case ST_TSO_E :
      case ST_TSOI_E :
        pin->pin_type |= l_axubPortTypeScanOut;
        break;
      case ST_TSC_E :
      case ST_TSCA_E :
      case ST_TSCB_E :
        pin->pin_type |= l_axubPortTypeScanClock;
        break;
```

```
      case ST_TSE_E :
      case ST_TSEI_E :
      /* ---- no test_scan_enable is used in xxx (01/20/99) */
      /* pin->pin_type |= l_axubPortTypeEnable; */
      break;
      case ST_TCLK_E :
      break;
    }
  }
}

/* ==================================================== */
```

```
/*=================================================

   libc_def.c -- define value handle


 ================================================= */

#include "libc_def.h"

#define HASH_SIZE 29

public
int libc_def_init(void)
{
  def_table = new_libc_def_table_rec();
  def_table->gc_entry   = (struct libc_glb_const_rec
**)get_ptr_buffer(HASH_SIZE);
  def_table->lib  = (struct libc_def_entry_rec **)get_ptr_buffer(HASH_SIZE);
  def_table->cell = (struct libc_def_entry_rec **)get_ptr_buffer(HASH_SIZE);
  def_table->pin  = (struct libc_def_entry_rec **)get_ptr_buffer(HASH_SIZE);
}

/* ================================================= */

  static
  int libc_def_hash_func(
        text_buffer *name)
  { char *s;
    unsigned int v = 0;

    for (s=name;(*s)!='\0';s++)
      v += (*s);

    v %= HASH_SIZE;

    return(v);
  }

/* ------------------------------------------------- */

public
void libc_def_insert(
      text_buffer *def_name,
      text_buffer *region_name,     /* library, cell, pin */
      text_buffer *type_name)
{ unsigned int idx;
  libc_def_entry_rec *p;
  libc_def_entry_rec **entry_head;
  char msg[256];
  enum value_type vt;

  if (strcmp(region_name,"library") == 0)
    entry_head = def_table->lib;
  else if (strcmp(region_name,"cell") == 0)
    entry_head = def_table->cell;
  else if (strcmp(region_name,"pin") == 0)
```

```
    entry_head = def_table->pin;
  else {
    sprintf(msg,"Not support user define in group %s.",region_name);
    libcerror(msg);
  }
  free_text_buffer(region_name);

  if (strcmp(type_name,"integer")==0 || strcmp(type_name,"float")==0)
    vt = REAL_VT;
  else if (strcmp(type_name,"string")==0)
    vt = TEXT_VT;
  else if (strcmp(type_name,"boolean")==0)
    vt = BOOL_VT;
  else {
    sprintf(msg,"Not support user define type %s.",type_name);
    libcerror(msg);
  }
  free_text_buffer(type_name);

  if (entry_head == NULL) {
    free_text_buffer(def_name);
    exit(44);
  }

  idx = libc_def_hash_func(def_name);
  for (p=entry_head[idx];p!=NULL;p=p->next) {
    if (strcmp(p->def_name,def_name) == 0) {          /* find it, overwrite
*/
      free_text_buffer(def_name);
      p->v_type = vt;
      return;
    }
  }
  p          = new_libc_def_entry_rec();
  p->next    = entry_head[idx];
  p->def_name      = def_name;
  p->v_type        = vt;
  entry_head[idx] = p;
}

/* ------------------------------------------------ */

public
void libc_def_cell_area(
      text_buffer *def_name,
      enum pad_type_E resource_type)
{ text_buffer *area_name,*region_name,*type_name;
  libc_cell_area_rec *cap;

  area_name   = copy_string(def_name);
  region_name = copy_string("cell");
  type_name   = copy_string("float");
  libc_def_insert(def_name,region_name,type_name);
  cap = new_libc_cell_area_rec();
  cap->area_name = area_name;
  cap->resource_type = resource_type;
  cap->next = tech_lib->define_cell_area;
```

```
  tech_lib->define_cell_area = cap;
}


/* ------------------------------------------------ */

public
int libc_def_find(
      text_buffer *name,
      enum value_type *vt)
{ unsigned int idx;
  libc_def_entry_rec *p;
  libc_def_entry_rec **entry_head;

  /* ---- find the region first */
  if (lib_pin != NULL)
    entry_head = def_table->pin;
  else if (lib_cell != NULL)
    entry_head = def_table->cell;
  else
    entry_head = def_table->lib;

  idx = libc_def_hash_func(name);
  for (p=entry_head[idx];p!=NULL;p=p->next) {
    if (strcmp(p->def_name,name) == 0) {                /* found */
      (*vt) = p->v_type;
      return(1);        /* SUCC */
    }
  }
  return(0);                  /* FAIL (not found) */
}

/* ================================================ */

public
void libc_def_gc_insert(
      text_buffer *name,
      float value)
{ int idx;
  libc_glb_const_rec *p;

  idx = libc_def_hash_func(name);
  for (p=def_table->gc_entry[idx];p!=NULL;p=p->next) {
    if (strcmp(p->gc_name,name) == 0) {             /* find it, overwrite */
      free_text_buffer(name);
      p->value = value;
      return;
    }
  }
  p          = new_libc_glb_const_rec();
  p->next    = def_table->gc_entry[idx];
  p->gc_name       = name;
  p->value         = value;
  def_table->gc_entry[idx] = p;

}
```

```
/* ------------------------------------------------- */

public
int libc_def_gc_find(
        text_buffer *name,
        float *value)
{ unsigned int idx;
  libc_glb_const_rec *p;
  int code = 0;

  idx = libc_def_hash_func(name);
  for (p=def_table->gc_entry[idx];p!=NULL;p=p->next) {
    if (strcmp(p->gc_name,name) == 0) {                 /* found */
      free_text_buffer(name);
      (*value) = p->value;
      return(1);                /* 1: SUCC */
    }
  }
  if (strcmp(name,"VDD") == 0) {
    (*value) = 3.3;
    code = 1;
  }
  free_text_buffer(name);
  (*value) = 1.0;
  return(code);                /* 0: FAIL (not found) */
}

/* ================================================= */
```

```
/*================================================


 ================================================ */

#ifndef LIBC_DEF_H      /* { */
#define LIBC_DEF_H

#include <stdio.h>
#include <string.h>
#include <assert.h>

#include "libc_mem.h"

/* ---- ref axu/axuExt.h */
#define l_axubPortTypeIn          (0x1)
#define l_axubPortTypeOut         (1<<1)
#define l_axubPortTypeIO          (1<<2)
#define l_axubPortTypeTri         (1<<3)
#define l_axubPortTypePower       (1<<4)
#define l_axubPortTypeGround      (1<<5)
#define l_axubPortTypeClock       (1<<6)
#define l_axubPortTypeTieUp       (1<<7)
#define l_axubPortTypeTieDn       (1<<8)
#define l_axubPortTypeAsyncRise   (1<<9)
#define l_axubPortTypeAsyncFall   (1<<10)
#define l_axubPortTypeDataIn      (1<<11)
#define l_axubPortTypeDataOut     (1<<12)
#define l_axubPortTypeAddressIn   (1<<13)
#define l_axubPortTypeEnable      (1<<14)
#define l_axubPortTypeScanIn      (1<<15)
#define l_axubPortTypeScanOut     (1<<16)
#define l_axubPortTypeScanClock   (1<<17)
#define l_axubPortTypeTriDisable (1<<18)

#ifndef public
#define public
#endif

/* ------------------------------------------------ */

/* sematic rec ------------------------------------ */

union yystype {         /* sematic record */
  int                   int_val;
  float                       real_val;
  text_buffer                 *string;

  struct {
    struct libc_name_list_rec *head;
    struct libc_name_list_rec *_tail;
  } name_list;

  struct {
    struct libc_float_list_rec      *head;
    struct libc_float_list_rec      *_tail;
```

```
  } float_list;

  struct {
    struct libc_float_list_list     *head;
    struct libc_float_list_list     *_tail;
  } float_list_list;

  struct any_unit_rec         *unit;

  struct libc_cell_rec        *cell;

  enum pad_type_E       pad_type;
  enum variable_E       var_type;
  enum dont_false_E           sa_type;
  enum signal_type_E          sig_type;

  struct libc_oc_power_rail_rec *prp;

  struct libc_bool_opr_rec    *bool_opr;

  struct libc_timing_rec      *timing;

};
#define YYSTYPE union yystype

/* ================================================ */

#ifndef LIBC_MEM
#define LIBC_MEM  extern
#define LIBC_INIT(s)
#else
#define LIBC_INIT(s)    = s
#endif

LIBC_MEM int libc_version LIBC_INIT(330);

LIBC_MEM int libc_ignore_flag LIBC_INIT(0);
LIBC_MEM int libc_error_count LIBC_INIT(0);

LIBC_MEM FILE *Aclf LIBC_INIT(NULL);            /* xxx clf file */
LIBC_MEM FILE *Sclf LIBC_INIT(NULL);            /* xxx clf file */
LIBC_MEM FILE *Sclf1 LIBC_INIT(NULL);           /* xxx con file for version
3.2 */
LIBC_MEM FILE *Mclf LIBC_INIT(NULL);            /* (power) zzz clf file */

/* ---- value for process */
LIBC_MEM float libc_p_nom LIBC_INIT(1.0); /* normal process */
LIBC_MEM float libc_t_nom LIBC_INIT(25.0);      /* normal temp */
LIBC_MEM float libc_v_nom LIBC_INIT(3.3); /* normal volt */
LIBC_MEM float libc_p_min LIBC_INIT(0.8); /* min process */
LIBC_MEM float libc_t_min LIBC_INIT(0.0); /* min temp */
LIBC_MEM float libc_v_min LIBC_INIT(2.9); /* min volt */
LIBC_MEM float libc_p_max LIBC_INIT(1.2); /* max process */
LIBC_MEM float libc_t_max LIBC_INIT(85.0);      /* max temp */
LIBC_MEM float libc_v_max LIBC_INIT(3.6); /* max volt */

LIBC_MEM float cap_scale LIBC_INIT(1.0);
```

```
LIBC_MEM float resist_scale LIBC_INIT(1.0);
LIBC_MEM float time_scale LIBC_INIT(1.0);
LIBC_MEM float watt_scale LIBC_INIT(1.0);
LIBC_MEM float joule_scale LIBC_INIT(1.0);

/* ------------------------------------------------ */

LIBC_MEM struct libc_lib_rec *tech_lib LIBC_INIT(NULL);
LIBC_MEM struct libc_k_factor_rec *kfc LIBC_INIT(NULL);
LIBC_MEM struct libc_k_factor_rec *sc_kfc LIBC_INIT(NULL);  /*
scaling_factors group */

LIBC_MEM struct libc_def_table_rec *def_table LIBC_INIT(NULL);
LIBC_MEM struct libc_lu_table_template_rec *lu_table_temp LIBC_INIT(NULL);
LIBC_MEM struct libc_lu_table_template_rec *plut_temp LIBC_INIT(NULL);
LIBC_MEM struct libc_operating_condition_rec *lib_op_cond LIBC_INIT(NULL);
LIBC_MEM struct libc_oc_power_rail_rec *lib_prp LIBC_INIT(NULL);
LIBC_MEM struct libc_power_supply_rec *lib_ps LIBC_INIT(NULL);
LIBC_MEM struct libc_timing_range_rec  *lib_timing_range LIBC_INIT(NULL);
LIBC_MEM struct libc_type_rec  *lib_type_name LIBC_INIT(NULL);
LIBC_MEM struct libc_wire_load_rec *lib_wire_load LIBC_INIT(NULL);
LIBC_MEM struct libc_wire_load_selection_rec *lib_wire_load_sel
LIBC_INIT(NULL);

LIBC_MEM struct libc_cell_rec *lib_cell LIBC_INIT(NULL);
LIBC_MEM struct libc_pin_rec *lib_bundle LIBC_INIT(NULL);
LIBC_MEM struct libc_pin_rec *lib_bus LIBC_INIT(NULL);
LIBC_MEM struct libc_pin_rec *lib_pin LIBC_INIT(NULL);

/* ---- test_cell */
LIBC_MEM struct libc_cell_rec *lib_tc_cell LIBC_INIT(NULL);
LIBC_MEM struct libc_name_list_rec *lib_tc_pin_name LIBC_INIT(NULL);

LIBC_MEM struct libc_ff_latch_rec *lib_ff_latch LIBC_INIT(NULL);
LIBC_MEM struct libc_internal_power *lib_int_power LIBC_INIT(NULL);
LIBC_MEM struct libc_leakage_power *lib_lkg_power LIBC_INIT(NULL);
LIBC_MEM struct libc_memory_rec *lib_memory LIBC_INIT(NULL);
LIBC_MEM struct libc_routing_track_rec *lib_routing_track LIBC_INIT(NULL);

LIBC_MEM struct libc_timing_rec *lib_timing LIBC_INIT(NULL);
LIBC_MEM struct libc_table_val_rec *lib_timing_tbl LIBC_INIT(NULL);

LIBC_MEM struct libc_min_pulse_width_rec *lib_mpw LIBC_INIT(NULL);
LIBC_MEM struct libc_minimum_period_rec *lib_mp LIBC_INIT(NULL);

LIBC_MEM struct libc_memory_write_rec *lib_memory_write LIBC_INIT(NULL);

/* ------------------------------------------------ */

/* ================================================ */
/* ---- libc_cell.c */

libc_pin_rec *libc_cell_find_pin_by_name(libc_cell_rec *,char *,int);
int libc_cell_get_bundle_idx(libc_cell_rec *,char *);
libc_name_list_rec *libc_cell_bus_name(libc_pin_rec *);
libc_k_factor_rec *libc_cell_k_factor(void);
libc_k_factor_rec *libc_cell_find_sc_group(text_buffer *);
```

```
/* ------------------------------------------------ */
/* ---- libc_def.c */

void libc_def_insert(text_buffer *,text_buffer *,text_buffer *);

/* ------------------------------------------------ */
/* ---- libc_opr.c */

libc_bool_opr_rec *libc_opr_handle(libc_bool_type_E,int);
void libc_opr_print(FILE *,libc_bool_opr_rec *,libc_cell_rec *,int);

/* ------------------------------------------------ */
/* ---- libc_time.c */

float_buffer *libc_time_copy_float_buf(float_buffer *);
int libc_time_find_template(text_buffer *,libc_lu_table_template_rec
*,libc_table_val_rec *);
void libc_time_handle(int);
void libc_time_minimum_period(int,float);


/* ------------------------------------------------ */

/* ------------------------------------------------ */
/* ---- libc_util.c */

any_unit_rec *libc_util_unit(float);
float_buffer *libc_util_float_list2buffer(libc_float_list_rec *);
float_buffer **libc_util_float_lists2buffer(libc_float_list_list *);
float_buffer *libc_util_float_lists3buffer(int,int,int,libc_float_list_list
*);
libc_bool_opr_rec *libc_util_bool_opr(libc_bool_type_E,int);

void libc_util_wire_load_fanout(int,float,float,float,int);
void libc_util_wire_load_table(int,int,float);
void libc_util_wl_select(float,float,text_buffer *);

/* ================================================ */

#endif                  /* } */
```

```
/*================================================


 ================================================ */

#include "libc_def.h"

#define TRI_TBL_NO      70

/* ================================================ */

/* ---- k factor */

static
void libc_gen_k_factor(
      libc_k_factor_rec *kfc)
{ int i;
  char *sc_type[3];

  sc_type[0] = "process";
  sc_type[1] = "temp";
  sc_type[2] = "volt";

  if (kfc->kf_name != NULL)
    fprintf(Aclf,"; ==================== k_factor name : %s ;\n",kfc-
>kf_name);
  else
    fprintf(Aclf,"; ======================================= k_factor
====\n");
  for (i=0;i<3;i++) {
    if (kfc->k_cell_rise[i] != 0)
      fprintf(Aclf,"; k_%s_cell_rise : %G ;\n",sc_type[i],kfc-
>k_cell_rise[i]);
    if (kfc->k_cell_fall[i] != 0)
      fprintf(Aclf,"; k_%s_cell_fall : %G ;\n",sc_type[i],kfc-
>k_cell_fall[i]);
    if (kfc->k_cell_leakage_power[i] != 0)
      fprintf(Aclf,"; k_%s_cell_leakage_power : %G ;\n",sc_type[i],kfc-
>k_cell_leakage_power[i]);
    if (kfc->k_cell_power[i] != 0)
      fprintf(Aclf,"; k_%s_cell_power : %G ;\n",sc_type[i],kfc-
>k_cell_power[i]);
    if (kfc->k_drive_fall[i] != 0)
      fprintf(Aclf,"; k_%s_drive_fall : %G ;\n",sc_type[i],kfc-
>k_drive_fall[i]);
    if (kfc->k_drive_rise[i] != 0)
      fprintf(Aclf,"; k_%s_drive_rise : %G ;\n",sc_type[i],kfc-
>k_drive_rise[i]);
    if (kfc->k_fall_delay_intercept[i] != 0)
      fprintf(Aclf,"; k_%s_fall_delay_intercept : %G ;\n",sc_type[i],kfc-
>k_fall_delay_intercept[i]);
    if (kfc->k_fall_pin_resistance[i] != 0)
      fprintf(Aclf,"; k_%s_fall_pin_resistance : %G ;\n",sc_type[i],kfc-
>k_fall_pin_resistance[i]);
    if (kfc->k_fall_propagation[i] != 0)
```

```
      fprintf(Aclf,"; k_%s_fall_propagation : %G ;\n",sc_type[i],kfc-
>k_fall_propagation[i]);
    if (kfc->k_fall_transition[i] != 0)
      fprintf(Aclf,"; k_%s_fall_transition : %G ;\n",sc_type[i],kfc-
>k_fall_transition[i]);
    if (kfc->k_fall_wire_resistance[i] != 0)
      fprintf(Aclf,"; k_%s_fall_wire_resistance : %G ;\n",sc_type[i],kfc-
>k_fall_wire_resistance[i]);
    if (kfc->k_fall_wor_emitter[i] != 0)
      fprintf(Aclf,"; k_%s_fall_wor_emitter : %G ;\n",sc_type[i],kfc-
>k_fall_wor_emitter[i]);
    if (kfc->k_fall_wor_intercept[i] != 0)
      fprintf(Aclf,"; k_%s_fall_wor_intercept : %G ;\n",sc_type[i],kfc-
>k_fall_wor_intercept[i]);
    if (kfc->k_hold_fall[i] != 0)
      fprintf(Aclf,"; k_%s_hold_fall : %G ;\n",sc_type[i],kfc-
>k_hold_fall[i]);
    if (kfc->k_hold_rise[i] != 0)
      fprintf(Aclf,"; k_%s_hold_rise : %G ;\n",sc_type[i],kfc-
>k_hold_rise[i]);
    if (kfc->k_internal_power[i] != 0)
      fprintf(Aclf,"; k_%s_internal_power : %G ;\n",sc_type[i],kfc-
>k_internal_power[i]);
    if (kfc->k_intrinsic_fall[i] != 0)
      fprintf(Aclf,"; k_%s_intrinsic_fall : %G ;\n",sc_type[i],kfc-
>k_intrinsic_fall[i]);
    if (kfc->k_intrinsic_rise[i] != 0)
      fprintf(Aclf,"; k_%s_intrinsic_rise : %G ;\n",sc_type[i],kfc-
>k_intrinsic_rise[i]);
    if (kfc->k_min_period[i] != 0)
      fprintf(Aclf,"; k_%s_min_period : %G ;\n",sc_type[i],kfc-
>k_min_period[i]);
    if (kfc->k_min_pulse_width_high[i] != 0)
      fprintf(Aclf,"; k_%s_min_pulse_width_high : %G ;\n",sc_type[i],kfc-
>k_min_pulse_width_high[i]);
    if (kfc->k_min_pulse_width_low[i] != 0)
      fprintf(Aclf,"; k_%s_min_pulse_width_low : %G ;\n",sc_type[i],kfc-
>k_min_pulse_width_low[i]);
    if (kfc->k_pin_cap[i] != 0)
      fprintf(Aclf,"; k_%s_pin_cap : %G ;\n",sc_type[i],kfc->k_pin_cap[i]);
    if (kfc->k_pin_power[i] != 0)
      fprintf(Aclf,"; k_%s_pin_power : %G ;\n",sc_type[i],kfc-
>k_pin_power[i]);
    if (kfc->k_recovery_fall[i] != 0)
      fprintf(Aclf,"; k_%s_recovery_fall : %G ;\n",sc_type[i],kfc-
>k_recovery_fall[i]);
    if (kfc->k_recovery_rise[i] != 0)
      fprintf(Aclf,"; k_%s_recovery_rise : %G ;\n",sc_type[i],kfc-
>k_recovery_rise[i]);
    if (kfc->k_rise_delay_intercept[i] != 0)
      fprintf(Aclf,"; k_%s_rise_delay_intercept : %G ;\n",sc_type[i],kfc-
>k_rise_delay_intercept[i]);
    if (kfc->k_rise_pin_resistance[i] != 0)
      fprintf(Aclf,"; k_%s_rise_pin_resistance : %G ;\n",sc_type[i],kfc-
>k_rise_pin_resistance[i]);
    if (kfc->k_rise_propagation[i] != 0)
```

```
      fprintf(Aclf,"; k_%s_rise_propagation : %G ;\n",sc_type[i],kfc-
>k_rise_propagation[i]);
    if (kfc->k_rise_transition[i] != 0)
      fprintf(Aclf,"; k_%s_rise_transition : %G ;\n",sc_type[i],kfc-
>k_rise_transition[i]);
    if (kfc->k_rise_wire_resistance[i] != 0)
      fprintf(Aclf,"; k_%s_rise_wire_resistance : %G ;\n",sc_type[i],kfc-
>k_rise_wire_resistance[i]);
    if (kfc->k_rise_wor_emitter[i] != 0)
      fprintf(Aclf,"; k_%s_rise_wor_emitter : %G ;\n",sc_type[i],kfc-
>k_rise_wor_emitter[i]);
    if (kfc->k_rise_wor_intercept[i] != 0)
      fprintf(Aclf,"; k_%s_rise_wor_intercept : %G ;\n",sc_type[i],kfc-
>k_rise_wor_intercept[i]);
    if (kfc->k_setup_fall[i] != 0)
      fprintf(Aclf,"; k_%s_setup_fall : %G ;\n",sc_type[i],kfc-
>k_setup_fall[i]);
    if (kfc->k_setup_rise[i] != 0)
      fprintf(Aclf,"; k_%s_setup_rise : %G ;\n",sc_type[i],kfc-
>k_setup_rise[i]);
    if (kfc->k_slope_fall[i] != 0)
      fprintf(Aclf,"; k_%s_slope_fall : %G ;\n",sc_type[i],kfc-
>k_slope_fall[i]);
    if (kfc->k_slope_rise[i] != 0)
      fprintf(Aclf,"; k_%s_slope_rise : %G ;\n",sc_type[i],kfc-
>k_slope_rise[i]);
    if (kfc->k_wire_cap[i] != 0)
      fprintf(Aclf,"; k_%s_wire_cap : %G ;\n",sc_type[i],kfc->k_wire_cap[i]);
    if (kfc->k_wire_res[i] != 0)
      fprintf(Aclf,"; k_%s_wire_res : %G ;\n",sc_type[i],kfc->k_wire_res[i]);
  }
  fprintf(Aclf,"; ============================================================\n");
}

/* ------------------------------------------------ */

static
void libc_gen_scaled_value(
      float *k_factor,
      float *min_scaled,      /* OUT */
      float *max_scaled)      /* OUT */
{
  (*min_scaled) = (1+(libc_p_min-libc_p_nom) * k_factor[0]) *
              (1+(libc_t_min-libc_t_nom) * k_factor[1]) *
              (1+(libc_v_max-libc_v_nom) * k_factor[2]);
  (*max_scaled) = (1+(libc_p_max-libc_p_nom) * k_factor[0]) *
              (1+(libc_t_max-libc_t_nom) * k_factor[1]) *
              (1+(libc_v_min-libc_v_nom) * k_factor[2]);
}

/* ================================================ */

static char *timing_table_name[8] =
{ "\"RiseCellDelayTable\"",    "\"FallCellDelayTable\"",
  "\"RisePropagationTable\"", "\"FallPropagationTable\"",
  "\"RiseTransitionTable\"",  "\"FallTransitionTable\"",
  "\"RiseConstraintTable\"",  "\"FallConstraintTable\"" };
```

```
/* ------------------------------------------------ */

  static
  void libc_gen_table_name(
        char *cell_name,
        char *from_pin,
        char *to_pin,
        char unate,
        int tbl_idx,
        int min_nom_max)            /* 0,1,2 */
  {

fprintf(Aclf,"\"%s:%s:%s%c%d:%d\"",cell_name,from_pin,to_pin,unate,tbl_idx,mi
n_nom_max);
  }

/* ------------------------------------------------ */

  static
  char *libc_gen_varE2str(
        enum variable_E varE)
  { switch (varE) {
      case INPUT_NET_TRANSITION :         return("\"InputNetTransition\"");
      /* used */
      case TOTAL_OUTPUT_NET_CAPACITANCE : return("\"OutputCapacitance\"");
      /* used */
      case OUTPUT_NET_LENGTH :
      return("\"OutputNetLength\"");
      case OUTPUT_NET_WIRE_CAP :          return("\"OutputNetWireCap\"");
      case OUTPUT_NET_PIN_CAP :
      return("\"OutputNetPinCap\"");

      case RELATED_OUT_TOTAL_OUTPUT_NET_CAP :
      return("\"RelatedOutTotalNetCap\"");
      case RELATED_OUT_OUTPUT_NET_LENGTH :
      return("\"RelatedOutputOutNetLength\"");
      case RELATED_OUT_OUTPUT_NET_WIRE_CAP :
      return("\"RelatedOutputOutNetWireCap\"");
      case RELATED_OUT_OUTPUT_NET_PIN_CAP :
      return("\"RelatedOutputOutNetPinCap\"");

      case CONSTRAINED_PIN_TRANSITION :
      return("\"ConstrainedPinTransition\"");   /* used */
      case RELATED_PIN_TRANSITION :       return("\"RelatedPinTransition\"");
      /* used */
      case OUTPUT_PIN_TRANSITION :        return("\"OutputPinTransition\"");
      case CONNECT_DELAY :                return("\"ConnectDelay\"");
    }
    return(NULL);       /* VARIABLE_E_NONE */
  }

/* ------------------------------------------------ */

  static
  char *libc_gen_timetype2str(
        libc_timing_type_E tt)
```

```
  { switch(tt) {
      case RISING_EDGE_T :          return("\"CLOCK_RISING\"");
      case FALLING_EDGE_T :         return("\"CLOCK_FALLING\"");
      case PRESET_T :               return("\"PRESET\"");
      case CLEAR_T :                return("\"CLEAR\"");
      case HOLD_RISING_T :          return("\"HOLD_RISING\"");
      case HOLD_FALLING_T :         return("\"HOLD_FALLING\"");
      case SETUP_RISING_T :         return("\"SETUP_RISING\"");
      case SETUP_FALLING_T :        return("\"SETUP_FALLING\"");
      case RECOVERY_RISING_T :            return("\"RECOVERY_RISING\"");
      case RECOVERY_FALLING_T :           return("\"RECOVERY_FALLING\"");
      case THREE_STATE_DISABLE_T :  return("\"DISABLE\"");
      case THREE_STATE_ENABLE_T :   return("\"PATH\"");
      case REMOVAL_RISING_T :       return("\"REMOVAL_RISING\"");
      case REMOVAL_FALLING_T :            return("\"REMOVAL_FALLING\"");
      case COMBINATIONAL_T :        return("\"PATH\"");

      case SKEW_RISING_T :          return("\"SKEW_RISING\"");    /* <-- no
use */
      case SKEW_FALLING_T :         return("\"SKEW_FALLING\"");   /* <-- no
use */
      case NON_SEQ_HOLD_RISING_T :  return("\"NONSEQ_HOLD_RISING\"");
      case NON_SEQ_HOLD_FALLING_T : return("\"NONSEQ_HOLD_FALLING\"");
      case NON_SEQ_SETUP_RISING_T : return("\"NONSEQ_SETUP_RISING\"");
      case NON_SEQ_SETUP_FALLING_T :      return("\"NONSEQ_SETUP_FALLING\"");
      case NOCHANGE_HIGH_HIGH_T :   return("\"NOCHANGE_HIGH_HIGH\"");
      case NOCHANGE_HIGH_LOW_T :    return("\"NOCHANGE_HIGH_LOW\"");
      case NOCHANGE_LOW_HIGH_T :    return("\"NOCHANGE_LOW_HIGH\"");
      case NOCHANGE_LOW_LOW_T :           return("\"NOCHANGE_LOW_LOW\"");
    }
    assert(0);
    return("");
  }

/* ------------------------------------------------ */

  static
  char *libc_gen_unate2str(
        libc_timing_sense_E ts)
  { switch (ts) {
      case POSITIVE_UNATE_E :       return("\"noninvert\"");
      case NEGATIVE_UNATE_E :       return("\"invert\"");
      default :                     return("\"nonunate\"");
    }
  }

/* ------------------------------------------------ */

  static
  char libc_gen_unate2char(
        libc_timing_sense_E ts)
  { switch (ts) {
      case POSITIVE_UNATE_E :       return('+');
      case NEGATIVE_UNATE_E :       return('-');
      default :                     return(':');
    }
  }
```

```
/* ------------------------------------------------ */

  static
  void libc_gen_get_k_factor_table(
        libc_timing_type_E t_type,
        libc_k_factor_rec *kfc,
        float **kf1,
        float **kf2)
  {
    switch (t_type) {
      case COMBINATIONAL_T :
      case RISING_EDGE_T :
      case FALLING_EDGE_T :
      case PRESET_T :
      case CLEAR_T :
      case THREE_STATE_DISABLE_T :
      case THREE_STATE_ENABLE_T :
        /* ---- unchange */
        /* (*kf1) = (float *) (kfc->k_intrinsic_rise); or (kfc->k_cell_rise);
*/
        /* (*kf2) = (float *) (kfc->k_intrinsic_fall); or (kfc->k_cell_fall);
*/
        break;
      case HOLD_RISING_T :
      case HOLD_FALLING_T :
      case NON_SEQ_HOLD_RISING_T :
      case NON_SEQ_HOLD_FALLING_T :
        (*kf1) = (float *) (kfc->k_hold_rise);
        (*kf2) = (float *) (kfc->k_hold_fall);
        break;
      case SETUP_RISING_T :
      case SETUP_FALLING_T :
      case NON_SEQ_SETUP_RISING_T :
      case NON_SEQ_SETUP_FALLING_T :
        (*kf1) = (float *) (kfc->k_setup_rise);
        (*kf2) = (float *) (kfc->k_setup_fall);
        break;
      case RECOVERY_RISING_T :
      case RECOVERY_FALLING_T :
        (*kf1) = (float *) (kfc->k_recovery_rise);
        (*kf2) = (float *) (kfc->k_recovery_fall);
        break;
      case REMOVAL_RISING_T :
      case REMOVAL_FALLING_T :
        (*kf1) = (float *) (kfc->k_removal_rise);
        (*kf2) = (float *) (kfc->k_removal_fall);
        break;
      case SKEW_RISING_T :
      case SKEW_FALLING_T :
        (*kf1) = (float *) (kfc->k_skew_rise);
        (*kf2) = (float *) (kfc->k_skew_fall);
        break;
      case NOCHANGE_HIGH_HIGH_T :
      case NOCHANGE_HIGH_LOW_T :
      case NOCHANGE_LOW_HIGH_T :
      case NOCHANGE_LOW_LOW_T :
```

```
        (*kf1) = (float *) (kfc->k_nochange_rise);
        (*kf2) = (float *) (kfc->k_nochange_fall);
        break;
    }
  }

/* ------------------------------------------------ */

  static
  void libc_gen_get_scaling_factor(
        int idx,
        libc_timing_type_E t_type,
        libc_k_factor_rec *kfc,
        float *min_scaled,
        float *max_scaled)
  { float *k_factor,*kf1,*kf2;

    if (idx >= 6 && idx < TRI_TBL_NO) {
      kf1 = (float *) (kfc->k_cell_rise);
      kf2 = (float *) (kfc->k_cell_fall);
      libc_gen_get_k_factor_table(t_type,kfc,&kf1,&kf2);
      k_factor = ((idx%2)==0)? kf1 : kf2;
    }
    else {
      switch((idx % 10)) {
        case 0 :  k_factor = (float *) (kfc->k_cell_rise);  break;
        case 1 :  k_factor = (float *) (kfc->k_cell_fall);  break;
        case 2 :  k_factor = (float *) (kfc->k_rise_propagation); break;
        case 3 :  k_factor = (float *) (kfc->k_fall_propagation); break;
        case 4 :  k_factor = (float *) (kfc->k_rise_transition);  break;
        case 5 :  k_factor = (float *) (kfc->k_fall_transition);  break;
      }
    }
    libc_gen_scaled_value(k_factor,min_scaled,max_scaled);
  }

/* ------------------------------------------------ */

  static
  void libc_gen_1D_array(
        FILE *fp,
        float_buffer *array,
        float k_factor)
  { int i,size;

    size = sizeof_float_buffer(array);
    assert(size > 0);
    fprintf(fp,"(");
    for (i=0;i<size;i++)
      fprintf(fp,"%G ",array[i]*k_factor);
    fprintf(fp,")");
  }

/* ------------------------------------------------ */

  static
  void libc_gen_timing_values(
```

```
        FILE *fp,
        int size1,
        int size2,
        int size3,
        float_buffer *array,
        float k_factor)
  { int i,j,k;

    if (size2 == 0) {          /* 1D */
      fprintf(fp,"\n  '(");
      for (i=0;i<size1;i++)
        fprintf(fp,"%G ",array[i] * k_factor);
    }
    else if (size3 == 0) {     /* 2D */
      for (i=0;i<size1;i++) {
        if (i == 0)
          fprintf(fp,"\n  '(");
        else
          fprintf(fp,"\n    ");
        for (j=0;j<size2;j++)
          fprintf(fp,"%G ",array[i*size2+j] * k_factor);
      }
    }
    else {                /* 3D */
      for (i=0;i<size1;i++) {
        if (i == 0)
          fprintf(fp,"\n  '(");
        else
          fprintf(fp,"\n    ");
        for (j=0;j<size2;j++) {
          for (k=0;k<size3;k++)
            fprintf(fp,"%G ",array[(i*size2+j)*size3+k] * k_factor);
        }
      }
    }
    fprintf(fp,")\n");
  }

/* ------------------------------------------------ */

    static
    float libc_gen_varE2scale(
          enum variable_E varE)
    { switch (varE) {
        case INPUT_NET_TRANSITION :               return(time_scale);
        case TOTAL_OUTPUT_NET_CAPACITANCE :       return(cap_scale);
        case OUTPUT_NET_LENGTH :                  return(1.0);
        case OUTPUT_NET_WIRE_CAP :                return(cap_scale);
        case OUTPUT_NET_PIN_CAP :                 return(cap_scale);

        case RELATED_OUT_TOTAL_OUTPUT_NET_CAP :   return(cap_scale);
        case RELATED_OUT_OUTPUT_NET_LENGTH :      return(1.0);
        case RELATED_OUT_OUTPUT_NET_WIRE_CAP :    return(cap_scale);
        case RELATED_OUT_OUTPUT_NET_PIN_CAP :     return(cap_scale);

        case CONSTRAINED_PIN_TRANSITION :         return(time_scale);
        case RELATED_PIN_TRANSITION :             return(time_scale);
```

```
        case OUTPUT_PIN_TRANSITION :              return(time_scale);
        case CONNECT_DELAY :                      return(time_scale);
      }
      return(1.0);
    }

/* ------------------------------------------------ */

  static
  void libc_gen_clf_create_one_table(
        char *cell_name,
        char *from_pin,
        char *to_pin,
        char unate,
        int tbl_idx,
        int min_nom_max,            /* min:0,nom:1,max:2 */
        float k_factor,
        libc_table_val_rec *table)
  { char *idx_type1,*idx_type2,*idx_type3;
    variable_E v1,v2,v3;
    int size1,size2,size3,tbx_name_idx;

    fprintf(Aclf,"(clfCreateTable ");
    libc_gen_table_name(cell_name,from_pin,to_pin,unate,tbl_idx,min_nom_max);
    if (tbl_idx >= TRI_TBL_NO)
      tbl_idx %= 10;
    if (tbl_idx > 6)
      tbx_name_idx = (tbl_idx%2)? 7 : 6;
    else
      tbx_name_idx = tbl_idx;
    fprintf(Aclf," %s\n  '(",timing_table_name[tbx_name_idx]);

    if (table == NULL) {                    /* <--- no value(table) */
      fprintf(Aclf,") '(0)\n)\n");
    }
    else if (table->_tbl == NULL) {         /* ---- scalar table */
      fprintf(Aclf,") '(%G)\n)\n",table->scalar_val * time_scale);
    }
    else {
      v1        = table->_tbl->variable_1;
      v2        = table->_tbl->variable_2;
      v3        = table->_tbl->variable_3;
      idx_type1 = libc_gen_varE2str(v1);
      idx_type2 = libc_gen_varE2str(v2);
      idx_type3 = libc_gen_varE2str(v3);

      size1 = sizeof_float_buffer(table->index1);
      fprintf(Aclf,"(%s ",idx_type1);
      libc_gen_1D_array(Aclf,table->index1,libc_gen_varE2scale(v1));
      if (idx_type2 != NULL) {
        size2 = sizeof_float_buffer(table->index2);
        fprintf(Aclf,")\n    (%s ",idx_type2);
        libc_gen_1D_array(Aclf,table->index2,libc_gen_varE2scale(v2));
        if (idx_type3 != NULL) {
          size3 = sizeof_float_buffer(table->index3);
          fprintf(Aclf,")\n    (%s ",idx_type3);
          libc_gen_1D_array(Aclf,table->index3,libc_gen_varE2scale(v3));
```

```
          fprintf(Aclf,"))");
        libc_gen_timing_values(Aclf,size1,size2,size3,table->values,k_factor
* time_scale);
        }
        else {
          fprintf(Aclf,"))");
        libc_gen_timing_values(Aclf,size1,size2,0,table->values,k_factor *
time_scale);
        }
      }
      else {
        fprintf(Aclf,"))");
      libc_gen_timing_values(Aclf,size1,0,0,table->values,k_factor *
time_scale);
      }
      fprintf(Aclf,")\n");
    }
  }

/* ------------------------------------------------ */

static
void libc_gen_clf_create_table(
      char *cell_name,
      char *from_pin,
      char *to_pin,
      char unate,
      int tbl_idx,
      libc_k_factor_rec *kfc,
        libc_timing_type_E t_type,
      libc_table_val_rec *table,
      int index_t[])                /* OUT for min_nom_max */
{ float min_scaled,max_scaled;

  libc_gen_get_scaling_factor(tbl_idx,t_type,kfc,&min_scaled,&max_scaled);
  index_t[0] = 0; /* min */
  index_t[1] = 1; /* nom */
  index_t[2] = 2; /* max */
  if (min_scaled == 1.0 || table == NULL)
    index_t[0] = 1;
  else

libc_gen_clf_create_one_table(cell_name,from_pin,to_pin,unate,tbl_idx,0,min_s
caled,table);

libc_gen_clf_create_one_table(cell_name,from_pin,to_pin,unate,tbl_idx,1,1.0,t
able);
  if (max_scaled == 1.0 || table == NULL)
    index_t[2] = 1;
  else

libc_gen_clf_create_one_table(cell_name,from_pin,to_pin,unate,tbl_idx,2,max_s
caled,table);
}

/* ------------------------------------------------ */
```

```
static
int libc_gen_tbx_idx(
      libc_timing_type_E t_type,
      int default_value)
{
  switch (t_type) {
    case HOLD_RISING_T :            return(6);
    case HOLD_FALLING_T :           return(8);
    case SETUP_RISING_T :           return(10);
    case SETUP_FALLING_T :          return(12);
    case RECOVERY_RISING_T :        return(20);
    case RECOVERY_FALLING_T :       return(22);
    case REMOVAL_RISING_T :         return(24);
    case REMOVAL_FALLING_T :        return(26);
    case SKEW_RISING_T :            return(30);
    case SKEW_FALLING_T :           return(32);
    case NON_SEQ_HOLD_RISING_T :    return(40);
    case NON_SEQ_HOLD_FALLING_T :   return(42);
    case NON_SEQ_SETUP_RISING_T :   return(44);
    case NON_SEQ_SETUP_FALLING_T :  return(46);
    case NOCHANGE_HIGH_HIGH_T :           return(50);
    case NOCHANGE_HIGH_LOW_T :            return(52);
    case NOCHANGE_LOW_HIGH_T :            return(54);
    case NOCHANGE_LOW_LOW_T :       return(56);
    default :                       return(default_value);
  }
}

/* ------------------------------------------------ */

libc_pin_rec *libc_cell_find_pin_by_name(libc_cell_rec *,char *,int);
libc_name_list_rec *libc_cell_bus_name(libc_pin_rec *);

static
void libc_gen_TLU_timing(
      libc_timing_rec *timing)
{ char *cell_name;
  libc_cell_rec *curr_cell;
  libc_pin_rec *from_pp,*to_pp;
  libc_name_list_rec *from_pin,*to_pin;
  libc_name_list_rec *np1,*next_np1,*np1p,*np2,*next_np2,*np2p;
  libc_name_list_rec *np1_head,*np2_head;
  libc_k_factor_rec *kfc;
  int rise_tbl[3],fall_tbl[3],rise_tbl1[3],fall_tbl1[3];
  libc_timing_type_E t_type;
  int i,j,tbl_idx,tri_idx;
  int iidx,jidx1,jidx2,oidx;
  char unate;

  curr_cell = timing->_current_pin->_current_cell;
  unate     = libc_gen_unate2char(timing->timing_sense);
  cell_name = curr_cell->cell_name;
  from_pin  = timing->related_pin;
  to_pin    = timing->_current_pin->pin_name;
  kfc       = (curr_cell->_scaling_factors)? curr_cell->_scaling_factors :
tech_lib->k_factor;
```

```
t_type    = timing->timing_type;

for (np2=to_pin;np2!=NULL;np2=np2->next) {
  oidx      = libc_cell_get_bundle_idx(curr_cell,np2->name);
  next_np2  = np2->next;
  np2->next = NULL;
  to_pp = libc_cell_find_pin_by_name(curr_cell,np2->name,-99999);
  np2_head = (to_pp->is_bus)? libc_cell_bus_name(to_pp) : np2;
  jidx2    = (to_pp->is_bus)? 0 : -99999;
  for (np2p=np2_head;np2p!=NULL;np2p=np2p->next) {

    for (np1=from_pin;np1!=NULL;np1=np1->next) {
      next_np1  = np1->next;
      np1->next = NULL;
      from_pp = libc_cell_find_pin_by_name(curr_cell,np1->name,-99999);
    if (from_pp != NULL) {
        np1_head = (from_pp->members != NULL)? from_pp->members :
                   (from_pp->is_bus)? libc_cell_bus_name(from_pp) : np1;
      jidx1    = (from_pp->members != NULL || from_pp->is_bus)? 0 : -99999;
      }
    else {
      np1_head = np1;
      jidx1    = -99999;
    }
      for (np1p=np1_head;np1p!=NULL;np1p=np1p->next) {
      if (jidx1 == -99999)        /* pin -> pin/bundle/bus */
        iidx = -99999;
        else if (oidx != -99999) {      /* bundle/bus -> bundle */
        if (jidx1 != oidx)
          goto next_jidx1;
        iidx = oidx;
      }
      else if (jidx2 != -99999) { /* bundle/bus -> bus */
        if (jidx1 != jidx2)
          goto next_jidx1;
        iidx = jidx1;
      }
      else                        /* bundle/bus -> pin */
        iidx = jidx1;

        switch (t_type) {
          case CLEAR_T :
            if (timing->timing_sense == POSITIVE_UNATE_E)
            from_pp->pin_type |= l_axubPortTypeAsyncFall;
            else
            from_pp->pin_type |= l_axubPortTypeAsyncRise;
            break;
          case PRESET_T :
            if (timing->timing_sense == POSITIVE_UNATE_E)
            from_pp->pin_type |= l_axubPortTypeAsyncRise;
            else
            from_pp->pin_type |= l_axubPortTypeAsyncFall;
            break;
          case THREE_STATE_DISABLE_T :
          case THREE_STATE_ENABLE_T :
            from_pp->pin_type |= l_axubPortTypeTriDisable;
            to_pp->pin_type   |= l_axubPortTypeTri;
```

```
            break;
          case RISING_EDGE_T :
          case FALLING_EDGE_T :
            to_pp->pin_type   |= l_axubPortTypeDataOut;
            break;
          case SETUP_RISING_T :
          case SETUP_FALLING_T :
          case NON_SEQ_SETUP_RISING_T :
          case NON_SEQ_SETUP_FALLING_T :
            to_pp->pin_type   |= l_axubPortTypeDataIn;
            from_pp->pin_type |= l_axubPortTypeClock;
            break;
        }

      tri_idx = 0;
        switch (t_type) {
          case COMBINATIONAL_T :
          case RISING_EDGE_T :
          case FALLING_EDGE_T :
          case PRESET_T :
          case CLEAR_T :
          case THREE_STATE_DISABLE_T :
          case THREE_STATE_ENABLE_T :
          if (t_type == THREE_STATE_DISABLE_T)
          tri_idx = TRI_TBL_NO;
          if (t_type == THREE_STATE_ENABLE_T)
          tri_idx = TRI_TBL_NO + 10;

            if (timing->cell_rise != NULL || timing->cell_fall != NULL) {
            tbl_idx = 0;
            libc_gen_clf_create_table(cell_name,np1p->name,np2p-
>name,unate,tri_idx+0,
                kfc,t_type,timing->cell_rise,rise_tbl);
            libc_gen_clf_create_table(cell_name,np1p->name,np2p-
>name,unate,tri_idx+1,
                kfc,t_type,timing->cell_fall,fall_tbl);
            }
            else if (timing->rise_propagation != NULL || timing-
>fall_propagation != NULL) {
            tbl_idx = 2;
            libc_gen_clf_create_table(cell_name,np1p->name,np2p-
>name,unate,tri_idx+2,
                kfc,t_type,timing->rise_propagation,rise_tbl);
            libc_gen_clf_create_table(cell_name,np1p->name,np2p-
>name,unate,tri_idx+3,
                kfc,t_type,timing->fall_propagation,fall_tbl);
            }
            if (timing->rise_transition != NULL || timing->fall_transition
!= NULL) {
            libc_gen_clf_create_table(cell_name,np1p->name,np2p-
>name,unate,tri_idx+4,
                kfc,t_type,timing->rise_transition,rise_tbl1);
            libc_gen_clf_create_table(cell_name,np1p->name,np2p-
>name,unate,tri_idx+5,
                kfc,t_type,timing->fall_transition,fall_tbl1);
            }
          break;
```

```
          /* default : */
            case HOLD_RISING_T :
            case HOLD_FALLING_T :
            case SETUP_RISING_T :
            case SETUP_FALLING_T :
            case RECOVERY_RISING_T :
            case RECOVERY_FALLING_T :
            case REMOVAL_RISING_T :
            case REMOVAL_FALLING_T :
            case SKEW_RISING_T :
            case SKEW_FALLING_T :
            case NON_SEQ_HOLD_RISING_T :
            case NON_SEQ_HOLD_FALLING_T :
            case NON_SEQ_SETUP_RISING_T :
            case NON_SEQ_SETUP_FALLING_T :
            case NOCHANGE_HIGH_HIGH_T :
            case NOCHANGE_HIGH_LOW_T :
            case NOCHANGE_LOW_HIGH_T :
            case NOCHANGE_LOW_LOW_T :
              if (timing->rise_constraint != NULL || timing->fall_constraint)
{
              tbl_idx = libc_gen_tbx_idx(t_type,10);
              libc_gen_clf_create_table(cell_name,np1p->name,np2p-
>name,unate,tbl_idx,
                  kfc,t_type,timing->rise_constraint,rise_tbl);
              libc_gen_clf_create_table(cell_name,np1p->name,np2p-
>name,unate,tbl_idx+1,
                  kfc,t_type,timing->fall_constraint,fall_tbl);
              }
            break;
          }

          /* ---- generate defineTimeTLU */
        if (t_type == SKEW_RISING_T || t_type == SKEW_FALLING_T) {
            fprintf(Aclf,"(defineCondTimingSkewTLU \"%s\" \"%s\" \"%s\"
\"%s\" \"%s\" \"",cell_name,
            np1p->name,(t_type==SKEW_RISING_T)?"RISE":"FALL",
            np2p->name,(t_type==SKEW_RISING_T)?"RISE":"FALL");
          libc_opr_print(Aclf,timing->when_start,curr_cell,iidx);
          fprintf(Aclf,"\" \"");
          libc_opr_print(Aclf,timing->when_end,curr_cell,iidx);
          fprintf(Aclf,"\"\n  '(");
            for (i=0;i<3;i++) {
              libc_gen_table_name(cell_name,np1p->name,np2p-
>name,unate,tbl_idx,rise_tbl[i]);
              fprintf(Aclf," ");
              libc_gen_table_name(cell_name,np1p->name,np2p-
>name,unate,tbl_idx+1,fall_tbl[i]);
              fprintf(Aclf,(i==2)?"))\n":" ");
            }
        }
        else {          /* ---- normal defineTimeTLU */
          if (timing->when != NULL || timing->when_start != NULL || timing-
>when_end != NULL) {
              fprintf(Aclf,"(defineCondTimeTLU \"%s\" \"%s\" \"%s\" %s %s\n
\"",cell_name,np1p->name,np2p->name,
```

```
              libc_gen_unate2str(timing-
>timing_sense),libc_gen_timetype2str(t_type));
            libc_opr_print(Aclf,timing->when,curr_cell,iidx);
            fprintf(Aclf,"\" \"");
            libc_opr_print(Aclf,timing->when_start,curr_cell,iidx);
            fprintf(Aclf,"\" \"");
            libc_opr_print(Aclf,timing->when_end,curr_cell,iidx);
            fprintf(Aclf,"\"\n");
          }
          else {
              fprintf(Aclf,"(defineTimeTLU \"%s\" \"%s\" \"%s\" %s
%s\n",cell_name,np1p->name,np2p->name,
              libc_gen_unate2str(timing-
>timing_sense),libc_gen_timetype2str(t_type));
            }

          fprintf(Aclf,"  '(");
            for (i=0;i<3;i++) {
              libc_gen_table_name(cell_name,np1p->name,np2p-
>name,unate,tri_idx+tbl_idx,rise_tbl[i]);
              fprintf(Aclf," ");
              libc_gen_table_name(cell_name,np1p->name,np2p-
>name,unate,tri_idx+tbl_idx+1,fall_tbl[i]);
              fprintf(Aclf,(i==2)?")\n  '(":" ");
            }
            if (timing->rise_transition != NULL || timing->fall_transition) {
              for (i=0;i<3;i++) {
                libc_gen_table_name(cell_name,np1p->name,np2p-
>name,unate,tri_idx+4,rise_tbl1[i]);
                fprintf(Aclf," ");
                libc_gen_table_name(cell_name,np1p->name,np2p-
>name,unate,tri_idx+5,fall_tbl1[i]);
                fprintf(Aclf,(i==2)?"))\n":" ");
              }
            }
            else
              fprintf(Aclf,"))\n");
        }
next_jidx1 :
        if (from_pp != NULL)
          jidx1 += (from_pp->members != NULL || from_pp->is_bus)? 1 : 0;
        }
        np1->next = next_np1;
        if (from_pp != NULL && from_pp->is_bus)
          free_libc_name_list_rec(np1_head);
      }
      if (to_pp->is_bus)
      jidx2++;
    }
    np2->next = next_np2;
    if (to_pp->is_bus)
      free_libc_name_list_rec(np2_head);
  }
}

/* ------------------------------------------------ */
/* ------------------------------------------------ */
```

```
/* pin(X)
 * bundle(A), A1,A2, bundle(B), B1,B2
 * bus(C) C[1],C[2], bus(D) D[1],D[2]
 *
 * timing   : A->B  ==> A1->B1,  A2->B2
 * function : B = A ==> B1 = A1, B2 = A2
 * timing   : X->B  ==> X->B1, X->B2
 * function : B = X ==> B1 = X, B2 = X
 * timing   : A->X  ==> A1->X,  A2->X
 * function : X = A ==> (shall not happen)
 *
 * timing   : C->D  ==> C[1]->D[1], C[2]->D[2]
 * function : D = C ==> D[1] = C[1], D[2] = C[2]
 * timing   : X->D  ==> X->D[1], X->D[2]
 * function : D = X ==> D[1] = X, D[2] = X
 * timing   : C->X  ==> C[1]->X, C[2]->X
 * function : X = C ==> (shall not happen)
 */

static
void libc_gen_linear_timing(
      libc_timing_rec *timing)
{ char *cell_name;
  libc_cell_rec *curr_cell;
  libc_pin_rec *from_pp,*to_pp;
  libc_name_list_rec *from_pin,*to_pin;
  libc_name_list_rec *np1,*next_np1,*np1p,*np2,*next_np2,*np2p;
  libc_name_list_rec *np1_head,*np2_head;
  libc_timing_type_E t_type;
  libc_k_factor_rec *kfc;
  float *k_factor,*kf1,*kf2,min_scaled,max_scaled;
  float int_rise[3],int_fall[3];
  float slope_rise[3],slope_fall[3];
  float res_rise[3],res_fall[3];
  int i,iidx,jidx1,jidx2,oidx;

  assert (timing->_current_pin->members == NULL);

  curr_cell = timing->_current_pin->_current_cell;
  cell_name = curr_cell->cell_name;
  from_pin  = timing->related_pin;
  to_pin    = timing->_current_pin->pin_name;
  kfc       = (curr_cell->_scaling_factors)? curr_cell->_scaling_factors :
tech_lib->k_factor;

  t_type    = timing->timing_type;

  kf1 = (float *)(kfc->k_intrinsic_rise);       /* default value */
  kf2 = (float *)(kfc->k_intrinsic_fall);       /* default value */
  libc_gen_get_k_factor_table(t_type,kfc,&kf1,&kf2);

  k_factor = kf1;
  libc_gen_scaled_value(k_factor,&min_scaled,&max_scaled);
  int_rise[0] = timing->intrinsic_rise * time_scale * min_scaled;
  int_rise[1] = timing->intrinsic_rise * time_scale;
  int_rise[2] = timing->intrinsic_rise * time_scale * max_scaled;
```

```
k_factor = kf2;
libc_gen_scaled_value(k_factor,&min_scaled,&max_scaled);
int_fall[0] = timing->intrinsic_fall * time_scale * min_scaled;
int_fall[1] = timing->intrinsic_fall * time_scale;
int_fall[2] = timing->intrinsic_fall * time_scale * max_scaled;

/* ---- add time_scale in here */
k_factor = (float *) (kfc->k_slope_rise);
libc_gen_scaled_value(k_factor,&min_scaled,&max_scaled);
slope_rise[0] = timing->slope_rise * min_scaled;
slope_rise[1] = timing->slope_rise ;
slope_rise[2] = timing->slope_rise * max_scaled;

/* ---- add time_scale in here */
k_factor = (float *) (kfc->k_slope_fall);
libc_gen_scaled_value(k_factor,&min_scaled,&max_scaled);
slope_fall[0] = timing->slope_fall * min_scaled;
slope_fall[1] = timing->slope_fall ;
slope_fall[2] = timing->slope_fall * max_scaled;

k_factor = (float *) (kfc->k_drive_rise);
libc_gen_scaled_value(k_factor,&min_scaled,&max_scaled);
res_rise[0] = timing->rise_resistance * resist_scale * min_scaled;
res_rise[1] = timing->rise_resistance * resist_scale;
res_rise[2] = timing->rise_resistance * resist_scale * max_scaled;

k_factor = (float *) (kfc->k_drive_fall);
libc_gen_scaled_value(k_factor,&min_scaled,&max_scaled);
res_fall[0] = timing->fall_resistance * resist_scale * min_scaled;
res_fall[1] = timing->fall_resistance * resist_scale;
res_fall[2] = timing->fall_resistance * resist_scale * max_scaled;

for (np2=to_pin;np2!=NULL;np2=np2->next) {
  oidx      = libc_cell_get_bundle_idx(curr_cell,np2->name);
  next_np2  = np2->next;
  np2->next = NULL;
  to_pp = libc_cell_find_pin_by_name(curr_cell,np2->name,-99999);
  np2_head = (to_pp->is_bus)? libc_cell_bus_name(to_pp) : np2;
  jidx2    = (to_pp->is_bus)? 0 : -99999;
  for (np2p=np2_head;np2p!=NULL;np2p=np2p->next) {

    for (np1=from_pin;np1!=NULL;np1=np1->next) {
      next_np1  = np1->next;
      np1->next = NULL;
      from_pp = libc_cell_find_pin_by_name(curr_cell,np1->name,-99999);
    if (from_pp != NULL) {
        np1_head = (from_pp->members != NULL)? from_pp->members :
                   (from_pp->is_bus)? libc_cell_bus_name(from_pp) : np1;
      jidx1    = (from_pp->members != NULL || from_pp->is_bus)? 0 : -99999;
      }
    else {
      np1_head = np1;
      jidx1    = -99999;
    }
      for (np1p=np1_head;np1p!=NULL;np1p=np1p->next) {
      if (jidx1 == -99999)        /* pin -> pin/bundle/bus */
        iidx = -99999;
```

```
          else if (oidx != -99999) {      /* bundle/bus -> bundle */
          if (jidx1 != oidx)
            goto next_jidx1;
          iidx = oidx;
        }
        else if (jidx2 != -99999) { /* bundle/bus -> bus */
          if (jidx1 != jidx2)
            goto next_jidx1;
          iidx = jidx1;
        }
        else                          /* bundle/bus -> pin */
          iidx = jidx1;

          switch (t_type) {
            case CLEAR_T :
              if (timing->timing_sense == POSITIVE_UNATE_E)
              from_pp->pin_type |= l_axubPortTypeAsyncFall;
              else
              from_pp->pin_type |= l_axubPortTypeAsyncRise;
              break;
            case PRESET_T :
              if (timing->timing_sense == POSITIVE_UNATE_E)
              from_pp->pin_type |= l_axubPortTypeAsyncRise;
              else
              from_pp->pin_type |= l_axubPortTypeAsyncFall;
              break;
            case THREE_STATE_DISABLE_T :
            case THREE_STATE_ENABLE_T :
              from_pp->pin_type |= l_axubPortTypeTriDisable;
              to_pp->pin_type   |= l_axubPortTypeTri;
              break;
            case RISING_EDGE_T :
            case FALLING_EDGE_T :
              to_pp->pin_type   |= l_axubPortTypeDataOut;
              break;
            case SETUP_RISING_T :
            case SETUP_FALLING_T :
            case NON_SEQ_SETUP_RISING_T :
            case NON_SEQ_SETUP_FALLING_T :
              to_pp->pin_type   |= l_axubPortTypeDataIn;
              from_pp->pin_type |= l_axubPortTypeClock;
              break;
          }

        if (t_type == SKEW_RISING_T || t_type == SKEW_FALLING_T) {
            fprintf(Aclf,"defineCondTimingSkew \"%s\" \"%s\" \"%s\" \"%s\"
\"%s\" \"",cell_name,
            np1p->name,(t_type==SKEW_RISING_T)?"RISE":"FALL",
            np2p->name,(t_type==SKEW_RISING_T)?"RISE":"FALL");
          libc_opr_print(Aclf,timing->when_start,curr_cell,iidx);
          fprintf(Aclf,"\" \"");
          libc_opr_print(Aclf,timing->when_end,curr_cell,iidx);
          fprintf(Aclf,"\"  '(%G)\n",int_rise[1]);
        }
        else {
          if (timing->when != NULL || timing->when_start != NULL || timing-
>when_end != NULL) {
```

```
                fprintf(Aclf,"defineCondTimeIntrinsic \"%s\" \"%s\" \"%s\" %s
%s \"",
                cell_name,np1p->name,np2p->name,
                  libc_gen_unate2str(timing-
>timing_sense),libc_gen_timetype2str(t_type));
              libc_opr_print(Aclf,timing->when,curr_cell,iidx);
              fprintf(Aclf,"\" \"");
              libc_opr_print(Aclf,timing->when_start,curr_cell,iidx);
              fprintf(Aclf,"\" \"");
              libc_opr_print(Aclf,timing->when_end,curr_cell,iidx);
              fprintf(Aclf,"\"\n");
            }
            else {
                fprintf(Aclf,"defineTimeIntrinsic \"%s\" \"%s\" \"%s\" %s
%s\n",
                cell_name,np1p->name,np2p->name,
                  libc_gen_unate2str(timing-
>timing_sense),libc_gen_timetype2str(t_type));
            }
              fprintf(Aclf,"  '(%G %G %G %G %G %G)",

int_rise[0],int_fall[0],int_rise[1],int_fall[1],int_rise[2],int_fall[2]);
              switch (t_type) {
                case COMBINATIONAL_T :
                case RISING_EDGE_T :
                case FALLING_EDGE_T :
                case PRESET_T :
                case CLEAR_T :
                case THREE_STATE_DISABLE_T :
                case THREE_STATE_ENABLE_T :
                  fprintf(Aclf,"\n  '(%G %G %G %G %G %G)\n",

slope_rise[0],slope_fall[0],slope_rise[1],slope_fall[1],slope_rise[2],slope_f
all[2]);
                  fprintf(Aclf,"  '(%G %G %G %G %G %G)\n",

res_rise[0],res_fall[0],res_rise[1],res_fall[1],res_rise[2],res_fall[2]);
                break;
              default :
                fprintf(Aclf," 0 0\n");
                break;
              }
          }
next_jidx1 :
            if (from_pp != NULL)
            jidx1 += (from_pp->members != NULL || from_pp->is_bus)? 1 : 0;
          }
          np1->next = next_np1;
          if (from_pp != NULL && from_pp->is_bus)
            free_libc_name_list_rec(np1_head);
        }
        if (to_pp->is_bus)
          jidx2++;
      }
      np2->next = next_np2;
      if (to_pp->is_bus)
        free_libc_name_list_rec(np2_head);
```

```
  }
}

/* ================================================ */
/* ---- min_pulse_width */

static
void libc_gen_min_pulse_width(
      libc_pin_rec *current_pin)
{ char *cell_name;
  libc_cell_rec *curr_cell;
  libc_name_list_rec *from_pin;
  libc_name_list_rec *np1,*next_np1,*np1p,*np1_head;
  libc_min_pulse_width_rec *mpw;
  libc_k_factor_rec *kfc;
  float *k_factor,min_scaled,max_scaled;
  float constraint_high[3],constraint_low[3];
  int iidx,jidx;

  mpw = current_pin->min_pulse_width;
  if (mpw == NULL)
    return;

  /* ---- no bundle pin is allowed here */
  assert(current_pin->members == NULL);

  curr_cell = current_pin->_current_cell;
  cell_name = curr_cell->cell_name;
  from_pin  = current_pin->pin_name;
  kfc       = (curr_cell->_scaling_factors)? curr_cell->_scaling_factors :
tech_lib->k_factor;

  k_factor = (float *)(kfc->k_min_pulse_width_high);
  libc_gen_scaled_value(k_factor,&min_scaled,&max_scaled);
  constraint_high[0] = mpw->constraint_high * min_scaled;
  constraint_high[1] = mpw->constraint_high;
  constraint_high[2] = mpw->constraint_high * max_scaled;

  k_factor = (float *)(kfc->k_min_pulse_width_low);
  libc_gen_scaled_value(k_factor,&min_scaled,&max_scaled);
  constraint_low[0] = mpw->constraint_low * min_scaled;
  constraint_low[1] = mpw->constraint_low;
  constraint_low[2] = mpw->constraint_low * max_scaled;

  for (np1=from_pin;np1!=NULL;np1=np1->next) {
    next_np1  = np1->next;
    np1->next = NULL;
    np1_head = (current_pin->is_bus)? libc_cell_bus_name(current_pin) : np1;
    iidx     = (current_pin->is_bus)? 0 : -99999;

    for (np1p=np1_head;np1p!=NULL;np1p=np1p->next) {
      /* ---- assign iidx as bundle idx if np->name is belong to bundle-
>members */
      jidx = libc_cell_get_bundle_idx(curr_cell,np1p->name);
      iidx = (jidx != -99999)? jidx : iidx;

      if (mpw->when != NULL) {
```

```
        fprintf(Aclf,"defineCondMinClkPulseWidth \"%s\" \"%s\" \"HIGH\"
\"",cell_name,np1p->name);
        libc_opr_print(Aclf,mpw->when,curr_cell,iidx);
      fprintf(Aclf,"\"  '(%G %G
%G)\n",constraint_high[0],constraint_high[1],constraint_high[2]);

        fprintf(Aclf,"defineCondMinClkPulseWidth \"%s\" \"%s\" \"LOW\"
\"",cell_name,np1p->name);
        libc_opr_print(Aclf,mpw->when,curr_cell,iidx);
      fprintf(Aclf,"\"  '(%G %G
%G)\n",constraint_low[0],constraint_low[1],constraint_low[2]);
      }
      else {
        fprintf(Aclf,"defineMinClkPulseWidth \"%s\" \"%s\" \"HIGH\" '(%G %G
%G)\n",cell_name,np1p->name,
          constraint_high[0],constraint_high[1],constraint_high[2]);
        fprintf(Aclf,"defineMinClkPulseWidth \"%s\" \"%s\" \"LOW\" '(%G %G
%G)\n",cell_name,np1p->name,
          constraint_low[0],constraint_low[1],constraint_low[2]);
      }
      if (current_pin->is_bus)
        iidx++;
    }
    np1->next = next_np1;
    if (current_pin->is_bus)
      free_libc_name_list_rec(np1_head);
  }
}

/* ================================================= */
/* ---- minimum_period */

static
void libc_gen_min_period(
      libc_pin_rec *current_pin)
{ char *cell_name;
  libc_cell_rec *curr_cell;
  libc_name_list_rec *from_pin;
  libc_name_list_rec *np1,*next_np1,*np1p,*np1_head;
  libc_minimum_period_rec *mp;
  libc_k_factor_rec *kfc;
  float *k_factor,min_scaled,max_scaled;
  float constraint[3];
  int iidx,jidx;

  mp = current_pin->minimum_period;
  if (mp == NULL)
    return;

  /* ---- no bundle pin is allowed here */
  assert(current_pin->members == NULL);

  curr_cell = current_pin->_current_cell;
  cell_name = curr_cell->cell_name;
  from_pin  = current_pin->pin_name;
  kfc       = (curr_cell->_scaling_factors)? curr_cell->_scaling_factors :
tech_lib->k_factor;
```

```
  k_factor = (float *) (kfc->k_min_period);
  libc_gen_scaled_value(k_factor,&min_scaled,&max_scaled);
  constraint[0] = mp->constraint* min_scaled;
  constraint[1] = mp->constraint;
  constraint[2] = mp->constraint* max_scaled;

  for (np1=from_pin;np1!=NULL;np1=np1->next) {
    next_np1  = np1->next;
    np1->next = NULL;
    np1_head = (current_pin->is_bus)? libc_cell_bus_name(current_pin) : np1;
    iidx     = (current_pin->is_bus)? 0 : -99999;

    for (np1p=np1_head;np1p!=NULL;np1p=np1p->next) {
      /* ---- assign iidx as bundle idx if np->name is belong to bundle-
>members */
      jidx = libc_cell_get_bundle_idx(curr_cell,np1p->name);
      iidx = (jidx != -99999)? jidx : iidx;

      if (mp->when != NULL) {
        fprintf(Aclf,"defineCondMinClkPeriod \"%s\" \"%s\"
\"",cell_name,np1p->name);
      libc_opr_print(Aclf,mp->when,curr_cell,iidx);
      fprintf(Aclf,"\"  '(%G %G
%G)\n",constraint[0],constraint[1],constraint[2]);
      }
      else {
        fprintf(Aclf,"defineMinClkPeriod \"%s\" \"%s\" '(%G %G
%G)\n",cell_name,np1p->name,
          constraint[0],constraint[1],constraint[2]);
      }
      if (current_pin->is_bus)
      iidx++;
    }
    np1->next = next_np1;
    if (current_pin->is_bus)
      free_libc_name_list_rec(np1_head);
  }
}

/* ================================================== */

static
void libc_gen_cell_timing(
      libc_cell_rec *cell)
{ libc_pin_rec *pp;
  libc_timing_rec *tp;

  for(pp=cell->pins;pp!=NULL;pp=pp->next) {
    if (pp->members)
      continue;
    for (tp=pp->timing;tp!=NULL;tp=tp->next) {
      if (tech_lib->delay_model == GENERIC_CMOS) {
        libc_gen_linear_timing(tp);
      }
      else if (tech_lib->delay_model == TABLE_LOOKUP) {
        libc_gen_TLU_timing(tp);
```

```
      }
    }
    libc_gen_min_pulse_width(pp);
    libc_gen_min_period(pp);
  }
}

/* ================================================= */
/* ================================================= */

#define l_axucMaxPortTypes    19

static char * libc_PortTypeTable[l_axucMaxPortTypes+1] = {
        "Input",
        "Output",
        "Inout",
        "Tristate",
        "Power",
        "Ground",
        "Clock",
        "Tieup",
        "Tiedown",
        "Async_rising",
        "Async_falling",
        "DataIn",
        "DataOut",
        "AddressIn",
        "Enable",
        "ScanIn",
        "ScanOut",
        "ScanClock",
        "TristateDisable",
        0 };

/* ------------------------------------------------- */

/* return 1 : is_filpflop, 0 : is not flip_flop */

  static
  int libc_gen_pin_port_type(
        libc_pin_rec *pin,
        libc_name_list_rec *np_head)
  { libc_name_list_rec *np;
    int i;

    /* ---- for pin only, no bus and bundle */
    for (np=np_head;np!=NULL;np=np->next) {
      fprintf(Aclf,"  ( \"%s\"",np->name);
      for (i=0;i<l_axucMaxPortTypes;i++) {
      if (pin->pin_type & (1<<i))
          fprintf(Aclf," \"%s\"",libc_PortTypeTable[i]);
      }
      fprintf(Aclf," )\n");
    }
    if (pin->pin_type & l_axubPortTypeClock)
      return(1);
    return(0);
```

```
  }

/* ------------------------------------------------ */

/* ---- must after LUT (timing) information */

static
void libc_gen_pin_info(
      libc_cell_rec *cell)
{ libc_pin_rec *pp;
  libc_k_factor_rec *kfc;
  float min_scaled,max_scaled,*k_factor;
  float min_cap,cap,max_cap;
  libc_name_list_rec *np,*np_head;
  char *cn;       /* cell_name */
  int isFlop = 0,isLatch = 0,isRam = 0,isRom = 0;

  cn = cell->cell_name;
  if (cell->ff_latch != NULL) {
    if (cell->ff_latch->is_ff)
      isFlop = 1;
    else if (!(cell->ff_latch->is_state))
      isLatch = 1;
  }
  if (cell->memory != NULL) {
    if (cell->memory->is_ram)
      isRam = 1;
    else
      isRom = 1;
  }

  /* ---- 1. dbSetCellPortTypes */
  fprintf(Aclf,"dbSetCellPortTypes (clfGetCLFLibName) \"%s\"  '(\n",cn);
  for (pp=cell->pins;pp!=NULL;pp=pp->next) {
    if (pp->members)
      continue;
    np_head = (pp->is_bus)? libc_cell_bus_name(pp) : pp->pin_name;
    isFlop |= libc_gen_pin_port_type(pp,np_head);
    if (pp->is_bus)
      free_libc_name_list_rec(np_head);
  }
  fprintf(Aclf,") #f\n");

  /* ---- 2. definePortCapacitance */
  kfc = (cell->_scaling_factors)? cell->_scaling_factors : tech_lib-
>k_factor;
  k_factor = (float *)(kfc->k_pin_cap);
  libc_gen_scaled_value(k_factor,&min_scaled,&max_scaled);

  for (pp=cell->pins;pp!=NULL;pp=pp->next) {
    if (pp->members)
      continue;

    cap     = pp->capacitance  * cap_scale;
    min_cap = cap * min_scaled * cap_scale;
    max_cap = cap * max_scaled * cap_scale;
```

```
    np_head = (pp->is_bus)? libc_cell_bus_name(pp) : pp->pin_name;

    for (np=np_head;np!=NULL;np=np->next) {
      fprintf(Aclf,"definePortCapacitance \"%s\" \"%s\" \"fall max\"
%G\n",cn,np->name,max_cap);
      fprintf(Aclf,"definePortCapacitance \"%s\" \"%s\" \"rise max\"
%G\n",cn,np->name,max_cap);
      fprintf(Aclf,"definePortCapacitance \"%s\" \"%s\" \"fall nom\"
%G\n",cn,np->name,cap);
      fprintf(Aclf,"definePortCapacitance \"%s\" \"%s\" \"rise nom\"
%G\n",cn,np->name,cap);
      fprintf(Aclf,"definePortCapacitance \"%s\" \"%s\" \"fall min\"
%G\n",cn,np->name,min_cap);
      fprintf(Aclf,"definePortCapacitance \"%s\" \"%s\" \"rise min\"
%G\n",cn,np->name,min_cap);
    }
    if (pp->is_bus)
      free_libc_name_list_rec(np_head);
  }

  /* ---- 3. dbSetLModelSubType */
  if(isFlop || isLatch || isRam || isRom) {
    char *lm_type;

    if (isFlop)
      lm_type = "flipflop";
    else if (isLatch)
      lm_type = "latch";
    else if (isRam)
      lm_type = "ram";
    else if (isRom)
      lm_type = "rom";
    fprintf(Aclf,"setq tmpCellId (dbOpenCell \"%s.%s\" \"w\")\n",cn,"TIM");
    fprintf(Aclf,"dbSetLModelSubType tmpCellId \"%s\"\n",lm_type);
    fprintf(Aclf,"; dbCloseCell unnecessary; CLF load has already opened cell
to load capacitance & will close \n");
  }
}

/* ================================================= */

static
void libc_gen_wire_load_model(
      libc_lib_rec *tlib)
{ libc_wire_load_rec *wlp;
  libc_fanout_length_rec *pp;

  libc_wire_load_selection_rec *wlsp;
  libc_wire_load_from_area_rec *ap;
  char *mode;

#if 1

  return;          /* skip in this version (08/24/98) */

#else
  /* ---- defineWireLoad */
```

```
  for (wlp=tlib->wire_load;wlp!=NULL;wlp=wlp->next) {
    fprintf(Sclf,"defineWireLoad \"%s\" '(%G",wlp->wl_name,wlp->slope);
    for (pp=wlp->fanout_length;pp!=NULL;pp=pp->next)
      fprintf(Sclf,"\n    (%d %G %G %G %G)",pp->fanout,pp->length,
      pp->capacitance * cap_scale,pp->resistance * resist_scale,pp->area);
    fprintf(Sclf,")\n");
  }

  /* ---- defineWireLoadSel */
  for (wlsp=tlib->wire_load_selection;wlsp!=NULL;wlsp=wlsp->next) {
    if (wlsp->wls_name != NULL)
      fprintf(Sclf,"defineWireLoadSel \"%s\" '(",wlsp->wls_name);
    else
      fprintf(Sclf,"defineWireLoadSel \"default_wire_load_sel\" '(");
    for (ap=wlsp->area_table;ap!=NULL;ap=ap->next)
      fprintf(Sclf,"\n    (%G %G \"%s\")",ap->min_area,ap->max_area,ap-
>_load_model->wl_name);
    fprintf(Sclf,")\n");
  }

  /* ---- defineDefaultWireLoad */
  if (tlib->default_wire_load != NULL)
    fprintf(Sclf,"defineDefaultWireLoad \"%s\"\n",tlib->default_wire_load);

  if (tlib->default_wire_load_selection != NULL)
    fprintf(Sclf,"defineDefaultWireLoadSel \"%s\"\n",tlib-
>default_wire_load_selection);
  else {
    if (tlib->wire_load_selection != NULL) {
      if (tlib->wire_load_selection->wls_name == NULL)
        fprintf(Sclf,"defineDefaultWireLoadSel \"default_wire_load_sel\"\n");
      else
        fprintf(Sclf,"defineDefaultWireLoadSel \"%s\"\n",tlib-
>wire_load_selection->wls_name);
    }
  }

  /* ---- defineDefaultWireLoadMode */
  switch (tlib->default_wire_load_mode) {
    case TOP_WL :       mode = "top";           break;
    case SEGMENTED_WL :       mode = "segmented";     break;
    case ENCLOSED_WL :        mode = "enclosed";      break;
  }
  fprintf(Sclf,"defineDefaultWireLoadMode \"%s\"\n\n",mode);
#endif
}

/* ------------------------------------------------ */

static
void libc_gen_cell_lclf(
      libc_cell_rec *cp)
{ char *cn;
  libc_pin_rec *pp,*bundle_p;
  libc_name_list_rec *np,*np_head;
  int is_ff=0,is_latch=0;
  libc_ff_latch_rec *ffp;
```

```
  int i,iidx = -99999,jidx;

  cn = cp->cell_name;

  if (libc_version >= 330) {
    if (cp->dont_touch || cp->pad_cell)
      fprintf(Sclf,"defineCellDontTouch \"%s\"\n",cn);                    /* --
-- dont touch */
    if (cp->dont_use || cp->pad_cell)
      fprintf(Sclf,"defineCellDontUse \"%s\"\n",cn);               /* ----
dont use */
    if (cp->cell_footprint != NULL)
      fprintf(Sclf,"defineCellFootPrint \"%s\" \"%s\"\n",cn,cp-
>cell_footprint);
  }

  /* ---- cell function */
  if (libc_version >= 330) {
    for (ffp=cp->ff_latch;ffp!=NULL;ffp=ffp->next) {
      if (ffp->is_ff)
        is_ff = 1;
      else if (!ffp->is_state)
        is_latch = 1;
      else {             /* ffp->is_state */
        if (ffp->clock_on != NULL || ffp->next_state != NULL)
          is_ff = 1;
        else
          is_latch = 1;
      }
      /* ---- 2. ff or latch function */
      for(i=0,iidx=-1;i<ffp->width;i++) {
        if (ffp->width == 1) {
          iidx = -99999;
          fprintf(Sclf,"%s \"%s\" \"%s\" \"%s\" \"",
            (is_ff)?"defineFlipFlopFunction":"defineLatchFunction",cn,ffp-
>Q_name,ffp->QN_name);
        }
        else {
          iidx += 1;
          fprintf(Sclf,"%s \"%s\" \"%s[%d]\" \"%s[%d]\" \"",
            (is_ff)?"defineFlipFlopFunction":"defineLatchFunction",cn,ffp-
>Q_name,iidx,ffp->QN_name,iidx);
        }
        if (is_ff) {
          libc_opr_print(Sclf,ffp->clock_on,cp,iidx);
          fprintf(Sclf,"\" \"");
          libc_opr_print(Sclf,ffp->next_state,cp,iidx);
        }
        else {
          libc_opr_print(Sclf,ffp->enable,cp,iidx);
          fprintf(Sclf,"\" \"");
          libc_opr_print(Sclf,ffp->data_in,cp,iidx);
        }
        fprintf(Sclf,"\" \"");
        libc_opr_print(Sclf,ffp->clear,cp,iidx);
        fprintf(Sclf,"\" \"");
        libc_opr_print(Sclf,ffp->preset,cp,iidx);
```

```
        fprintf(Sclf,"\" \"%c\" \"%c\" \"",ffp->clear_preset_var1,ffp-
>clear_preset_var2);
        libc_opr_print(Sclf,ffp->on_also,cp,iidx);
        fprintf(Sclf,"\"\n");
      }
    }
  }

  if (Sclf1 != NULL)                    /* lib_version == 320 */
    fprintf(Sclf1,"setq cellId (dbOpenCell \"%s.TIM\" \"w\")\n",cn);

  for (pp=cp->pins;pp!=NULL;pp=pp->next) {
    if (pp->members)
      continue;
    np_head = (pp->is_bus)? libc_cell_bus_name(pp) : pp->pin_name;
    iidx    = (pp->is_bus)? 0 : -99999;
    for (np=np_head;np!=NULL;np=np->next) {
      if (libc_version >= 330) {
        if (pp->fanout_load != 0)                                    /* ---- for
max_fanout */
        fprintf(Sclf,"defineCellFanoutLoad \"%s\" \"%s\" %G\n",cn,np-
>name,pp->fanout_load);
        if (pp->direction == OUTPUT_E || pp->direction == INOUT_E) {
          if (pp->max_fanout != 0)                                   /* ----
max_fanout */
          fprintf(Sclf,"defineCellMaxFanout \"%s\" \"%s\" %G\n",cn,np-
>name,pp->max_fanout);
          if (pp->max_transition != 0)                               /* ----
max_transition */
          fprintf(Sclf,"defineCellMaxTransition \"%s\" \"%s\" %G\n",cn,np-
>name,pp->max_transition);
          if (pp->max_capacitance != 0)                              /* ----
max_capacitance */
          fprintf(Sclf,"defineCellMaxCapacitance \"%s\" \"%s\" %G\n",cn,np-
>name,pp->max_capacitance);
        }
      }
      else if (Sclf1 != NULL) {           /* for version 320 */
        /* if (pp->fanout_load != 0)
         *  fprintf(Sclf1,"dbSetFanout \"%s\" %G\n",np->name,pp-
>fanout_load); */
        if (pp->direction == OUTPUT_E || pp->direction == INOUT_E) {
          if (pp->max_fanout != 0)
/* ---- max_fanout */
          fprintf(Sclf1,"dbSetMaxFanout cellId \"%s\" %G\n",np->name,pp-
>max_fanout);
          if (pp->max_transition != 0)                                /* --
-- max_transition */
          fprintf(Sclf1,"dbSetMaxTransition cellId \"%s\" %G\n",np->name,pp-
>max_transition);
          if (pp->max_capacitance != 0)                              /* --
-- max_capacitance */
          fprintf(Sclf1,"dbSetMaxCapacitance cellId \"%s\" %G\n",np->name,pp-
>max_capacitance);
        }
      }
```

```
      /* ---- pin function */
      if (pp->function == NULL)
        continue;
      /* ---- assign iidx as bundle idx if np->name is belong to bundle-
>members */
      jidx = libc_cell_get_bundle_idx(cp,np->name);
      iidx = (jidx != -99999)? jidx : iidx;
      if (pp->three_state != NULL) {
      /* ---- 1. three state */
      if (libc_version >= 330) {
        fprintf(Sclf,"defineTristateFunction \"%s\" \"%s\" \"",cn,np->name);
          libc_opr_print(Sclf,pp->three_state,cp,iidx);
        fprintf(Sclf,"\" \"");
          libc_opr_print(Sclf,pp->function,cp,iidx);
        fprintf(Sclf,"\"\n");
      }
      }
      else if (libc_version < 330 && cp->ff_latch != NULL)
      /* do nothing */ ;
      else {
      /* ---- 2. ff or latch function */
        /* ---- 3. combinational function */
        fprintf(Sclf,"defineBooleanFunction \"%s\" \"%s\" \"",cn,np->name);
        libc_opr_print(Sclf,pp->function,cp,iidx);
        fprintf(Sclf,"\"\n");
      }
      if (pp->is_bus)
        iidx++;
    }
    if (pp->is_bus)
      free_libc_name_list_rec(np_head);
  }
  fprintf(Sclf,"\n");

  if (Sclf1 != NULL)                /* lib_version == 320 */
    fprintf(Sclf1,"dbCloseCell cellId\n");
}

/* ================================================ */

/* ---- power clf */

static char *power_table_name[3] =
{ "\"SwitchingPowerTable\"", "\"InternalPowerTable\"",
"\"LeakagePowerTable\"" };

/* ------------------------------------------------ */

  static
  void libc_gen_power_table_name(
          char *cell_name,
          char *pin_name,
          int tidx,
        char rise_fall)       /* r,f */
  {
    if (rise_fall == 'r' || rise_fall == 'f')
      fprintf(Mclf,"\"%s:%s::%d%c\"",cell_name,pin_name,tidx,rise_fall);
```

```
    else
      fprintf(Mclf,"\"%s:%s::%d\"",cell_name,pin_name,tidx);
  }

/* ------------------------------------------------ */

  static
  void libc_gen_mclf_create_one_table(
        char *cell_name,
        char *pin_name,
        int tidx,
        int tbl_idx,                /* 0: output, 1: input */
        char rise_fall,       /* r,f,t */
        libc_table_val_rec *table)
  { char *idx_type1,*idx_type2,*idx_type3;
    variable_E v1,v2,v3;
    int size1,size2,size3;
    float k_factor = 1.0;

    fprintf(Mclf,"(clfCreateTable ");
    libc_gen_power_table_name(cell_name,pin_name,tidx,rise_fall);
    fprintf(Mclf," %s\n  '(",power_table_name[tbl_idx]);

    if (table == NULL) {                    /* <--- no value(table) */
      fprintf(Mclf,") '(0)\n)\n");
    }
    else if (table->_tbl == NULL) {         /* ---- scalar table */
      fprintf(Mclf,") '(%G)\n)\n",table->scalar_val * watt_scale);
    }
    else {
      v1        = table->_tbl->variable_1;
      v2        = table->_tbl->variable_2;
      v3        = table->_tbl->variable_3;
      idx_type1 = libc_gen_varE2str(v1);
      idx_type2 = libc_gen_varE2str(v2);
      idx_type3 = libc_gen_varE2str(v3);

      size1 = sizeof_float_buffer(table->index1);
      fprintf(Mclf,"(%s ",idx_type1);
      libc_gen_1D_array(Mclf,table->index1,libc_gen_varE2scale(v1));
      if (idx_type2 != NULL) {
        size2 = sizeof_float_buffer(table->index2);
        fprintf(Mclf,")\n    (%s ",idx_type2);
        libc_gen_1D_array(Mclf,table->index2,libc_gen_varE2scale(v2));
        if (idx_type3 != NULL) {
          size3 = sizeof_float_buffer(table->index3);
          fprintf(Mclf,")\n    (%s ",idx_type3);
          libc_gen_1D_array(Mclf,table->index3,libc_gen_varE2scale(v3));
          fprintf(Mclf,"))");
        libc_gen_timing_values(Mclf,size1,size2,size3,table->values,k_factor
* watt_scale);
        }
        else {
          fprintf(Mclf,"))");
        libc_gen_timing_values(Mclf,size1,size2,0,table->values,k_factor *
watt_scale);
        }
```

```
      }
      else {
        fprintf(Mclf,"))");
      libc_gen_timing_values(Mclf,size1,0,0,table->values,k_factor *
watt_scale);
      }
      fprintf(Mclf,")\n");
    }
  }

/* ------------------------------------------------ */

  static
  void libc_gen_define_power(
        libc_cell_rec *cell,
        char *pn,              /* pin_name */
        int is_in_pin,
        libc_internal_power *ipp,
        int tidx,
        int iidx,
        char rise_fall) /* r,f,t */
  { char *cn;
    libc_name_list_rec *np;
    libc_pin_rec *pp;

    cn = cell->cell_name;

    fprintf(Mclf,"%s \"%s\" \"",(is_in_pin)? "defineInternalPower" :
"defineSwitchingPower",cn);
    if (is_in_pin) {
      /* ---- input */
      fprintf(Mclf,"%s%c",pn,rise_fall);
      if (ipp->when) {
        /* ---- CPt*(S0+R0) */
        fprintf(Mclf,"*(");
        libc_opr_power_when_print(Mclf,ipp->when,cell,iidx);
        fprintf(Mclf,")");
      }
      fprintf(Mclf,"\" \"");
      /* ---- output */
      if (ipp->related_pin) {
        for (np=ipp->related_pin;np;np=np->next) {
        libc_opr_id_print(Mclf,np->name,cell,iidx);
        fprintf(Mclf,"t%s",(np->next!=NULL)?"+":"");
      }
      }
      else
        fprintf(Mclf,"-");
    }
    else {
      /* ---- input */
      if (ipp->related_pin == NULL && ipp->when == NULL)
        fprintf(Mclf,"-");
      else {
      if (ipp->related_pin) {
          fprintf(Mclf,"(");
          for (np=ipp->related_pin;np;np=np->next) {
```

```
            libc_opr_id_print(Mclf,np->name,cell,iidx);
            fprintf(Mclf,"t%s",(np->next!=NULL)?"+":"");
          }
          fprintf(Mclf,")%s",(ipp->when)?"*":"");
        }
          if (ipp->when) {
            fprintf(Mclf,"(");
            libc_opr_power_when_print(Mclf,ipp->when,cell,iidx);
            fprintf(Mclf,")");
          }
        }
        fprintf(Mclf,"\" \"");
        /* ---- output */
        fprintf(Mclf,"%s%c",pn,rise_fall);
        if (ipp->equal_or_opposite_output) {
        for (np=ipp->equal_or_opposite_output;np;np=np->next) {
            libc_opr_id_print(Mclf,np->name,cell,iidx);
          fprintf(Mclf,"%c%s",rise_fall,(np->next!=NULL)?"*":"");
        }
        }
      }
      fprintf(Mclf,"\" \"TLU\" ");
      libc_gen_power_table_name(cn,pn,tidx,rise_fall);
      fprintf(Mclf,"\n");
    }

/* ---------------------------------------------------- */
/* ---------------------------------------------------- */

    static
    void libc_gen_switching_name(
          libc_cell_rec *cell,
          libc_name_list_rec *np_head)
    { libc_pin_rec *pin;
      libc_name_list_rec *np1,*np2,*np2_head;

      for (np1=np_head;np1;np1=np1->next) {
        pin = libc_cell_find_pin_by_name(cell,np1->name,-99999);
      if (pin == NULL)
        fprintf(Mclf,"%st%s",np1->name,(np1->next)?"+":"");
      else {
        if (pin->members) {
          for (np2=pin->members;np2;np2=np2->next)
            fprintf(Mclf,"%st%s",np2->name,(np2->next||np1->next)?"+":"");
        }
        else if (pin->is_bus) {
          np2_head = libc_cell_bus_name(pin);
          for (np2=np2_head;np2;np2=np2->next)
            fprintf(Mclf,"%st%s",np2->name,(np2->next||np1->next)?"+":"");
          free_libc_name_list_rec(np2_head);
        }
        else
          fprintf(Mclf,"%st%s",np1->name,(np1->next)?"+":"");
        }
      }
    }
```

```
/* ------------------------------------------------ */

  static
  void libc_gen_define_power_97(
        libc_cell_rec *cell,
        int is_in_pin,
        libc_internal_power *ipp,
        int tidx)
  { char *cn;
    libc_name_list_rec *np;

    cn = cell->cell_name;

    fprintf(Mclf,"%s \"%s\" \"",(is_in_pin)? "defineInternalPower" :
"defineSwitchingPower",cn);
    if (is_in_pin) {
      assert(ipp->outputs == NULL);
      libc_gen_switching_name(cell,ipp->inputs);
      fprintf(Mclf,"\" \"-");
    }
    else {
      if (ipp->outputs == NULL)
        fprintf(Mclf,"-\" \"-");
      else {
      if (ipp->inputs != NULL)
        libc_gen_switching_name(cell,ipp->inputs);
      else
        fprintf(Mclf,"-");
        fprintf(Mclf,"\" \"");
        libc_gen_switching_name(cell,ipp->outputs);
      }
    }
    fprintf(Mclf,"\" \"TLU\" ");
    libc_gen_power_table_name(cn,"X",tidx,'t');
    fprintf(Mclf,"\n");
  }

/* ------------------------------------------------ */

static
void libc_gen_cell_mclf(
     libc_cell_rec *cp)
{ char *cn;
  libc_pin_rec *pp,*bundle_p;
  libc_name_list_rec *np,*np_head;
  int iidx = -99999,jidx,tidx=0;
  libc_internal_power *ipp;
  libc_leakage_power *lpp;
  int is_in_pin;

  cn = cp->cell_name;

  /* ---- 1. cell leakage power */
  for (lpp=cp->leakage_power;lpp;lpp=lpp->next) {
    fprintf(Mclf,"defineLeakagePower \"%s\" \"");
    libc_opr_power_when_print(Mclf,lpp->when,cp,-99999);
    fprintf(Mclf,"\" \"-\" \"Constant\" %G\n",cn,
```

```
      lpp->value*joule_scale);
  }
  if (cp->cell_leakage_power != 0.0)
    fprintf(Mclf,"defineLeakagePower \"%s\" \"-\" \"-\" \"Constant\"
%G\n",cn,
      cp->cell_leakage_power*joule_scale);

  /* ---- 2. old method internal power (97) */
  for (ipp=cp->internal_power_c;ipp!=NULL;ipp=ipp->next) {
    if (ipp->inputs == NULL && ipp->outputs == NULL) {
      libc_gen_mclf_create_one_table(cn,"X",tidx,0,'t',ipp->power);
      libc_gen_define_power_97(cp,0,ipp,tidx++);
    }
    else {
      if (ipp->outputs != NULL) {
        libc_gen_mclf_create_one_table(cn,"X",tidx,0,'t',ipp->power);
      libc_gen_define_power_97(cp,0,ipp,tidx++);
      }
      else if (ipp->inputs != NULL) {
        libc_gen_mclf_create_one_table(cn,"X",tidx,1,'t',ipp->power);
      libc_gen_define_power_97(cp,1,ipp,tidx++);
      }
    }
  }

  /* ---- 3. pin (internal/switching) power */
  for (pp=cp->pins;pp!=NULL;pp=pp->next) {
    if (pp->members)
      continue;
    is_in_pin = (pp->direction == INPUT_E || pp->direction == INTERNAL_E)? 1
: 0;
    for (ipp=pp->internal_power;ipp!=NULL;ipp=ipp->next,tidx++) {

      /* ---- 3.1 create pin power table */
      if (ipp->rise_power != NULL)
        libc_gen_mclf_create_one_table(cn,pp->pin_name-
>name,tidx,is_in_pin,'r',ipp->rise_power);
      if (ipp->fall_power != NULL)
        libc_gen_mclf_create_one_table(cn,pp->pin_name-
>name,tidx,is_in_pin,'f',ipp->fall_power);
      if (ipp->power != NULL)
        libc_gen_mclf_create_one_table(cn,pp->pin_name-
>name,tidx,is_in_pin,'t',ipp->power);

      np_head = (pp->is_bus)? libc_cell_bus_name(pp) : pp->pin_name;
      iidx    = (pp->is_bus)? 0 : -99999;
      for (np=np_head;np!=NULL;np=np->next) {
        jidx = libc_cell_get_bundle_idx(cp,np->name);
        iidx = (jidx != -99999)? jidx : iidx;

        /* ---- 3.2 generate define power */
      if (ipp->rise_power != NULL)
          libc_gen_define_power(cp,np->name,is_in_pin,ipp,tidx,iidx,'r');
        if (ipp->fall_power != NULL)
          libc_gen_define_power(cp,np->name,is_in_pin,ipp,tidx,iidx,'f');
        if (ipp->power != NULL)
          libc_gen_define_power(cp,np->name,is_in_pin,ipp,tidx,iidx,'t');
```

```
        if (pp->is_bus)
          iidx++;
      }
      if (pp->is_bus)
        free_libc_name_list_rec(np_head);
    }
  }
}

/* ================================================= */

public
void libc_gen_clf_main(
      FILE *xxx_out_p,
      FILE *yyy_out_p,
      FILE *yyy_out_con,
      FILE *zzz_out_p,
      libc_lib_rec *tlib)
{ libc_cell_rec *cp;

  if (tlib->delay_model != GENERIC_CMOS && tlib->delay_model != TABLE_LOOKUP)
    return;

  Aclf = xxx_out_p;
  Sclf = yyy_out_p;
  Sclf1 = yyy_out_con;  /* libc_version == 320 */
  Mclf = zzz_out_p;

  fprintf(Aclf,"; CLF translated from synopsys library format file\n");
  fprintf(Aclf,"; ==== Library : %s ====\n",tlib->lib_name);
  fprintf(Aclf,"; Temperature
(%G:%G:%G)\n",libc_t_min,libc_t_nom,libc_t_max);
  fprintf(Aclf,"; Voltage
(%G:%G:%G)\n",libc_v_min,libc_v_nom,libc_v_max);
  fprintf(Aclf,"; Process Factor
(%G:%G:%G)\n",libc_p_min,libc_p_nom,libc_p_max);
  fprintf(Aclf,"; Capacitance multiplier %G \n",cap_scale);
  fprintf(Aclf,"; Resistance  multiplier %G \n",resist_scale);
  fprintf(Aclf,"; Time        multiplier %G \n",time_scale);

  libc_gen_k_factor(tlib->k_factor);

  if (Sclf != NULL) {
    if (Sclf != Aclf)
      fprintf(Sclf,"; ==== Library : %s ====\n",tlib->lib_name);
    libc_gen_wire_load_model(tlib);
  }

  if (Sclf1 != NULL) {              /* libc_version == 320 */
    fprintf(Sclf1,"geOpenLib\n");
    fprintf(Sclf1,"setFormField \"Open Library\" \"Library Name\"
\"%s\"\n",tlib->lib_name);
    fprintf(Sclf1,"formOK \"Open Library\"\n\n");
  }

  for (cp=tlib->cells;cp!=NULL;cp=cp->next) {
```

```
    if (Aclf != NULL) {
      libc_gen_cell_timing(cp);
      libc_gen_pin_info(cp);
    }
    if (Sclf != NULL)
      libc_gen_cell_lclf(cp);
    if (Mclf != NULL)
      libc_gen_cell_mclf(cp);
  }

  if (Sclf1 != NULL)                /* libc_version == 320 */
    fprintf(Sclf1,"\ngeCloseLib\n");
}

/* ================================================== */
```

```
%{
/*=====================================================

 =====================================================*/

#include "libc_def.h"
#include "libc_yacc.h"        /* from libc.y */
#include <stdlib.h>

#define YY_SKIP_YYWRAP  1
#undef yywrap
#define yywrap()        1

#define echo      /* ECHO */
#define RN(s)     return(s)

void lex_error(char *);

static int libc_lineno = 1;
extern char *file_name;
static int pss[100];          /* prev_state_stack */
static int psc = 0;           /* prev_state_count */

int scaling_attr; /* 0:k_process, 1:k_temp, 2:k_volt */

#define NEXT_STATE      pss[psc++]=YY_START; BEGIN
#define PREV_STATE      BEGIN pss[--psc]

FILE *yyin;

%}

%s          PS LIB CU DM LU PU RU VU TU IPO
%s          BOOL MMP ANY_S WLM WLF TT SA DIR DRT NST
%s          PFT PSC BUS PIN CBO CM CIP PIP CELL IV LTT
%s          OV OC PC PLT TD SC SF TR TYPE WL
%s          WLS WLT PP FF CL MR MW TI MPW MP
%s          TIT TIS VS CTC RT PT CST CRT SIG_T CENUM
%s          OST SDF_E NAME QSTR LUT ROP REV CLP
%x          SKIP

ws          [ \b\t\v\f\r]+
integer           [0-9]+
real        ([0-9]+([.][0-9]*)?)([Ee][+-]?[0-9]+)?|([0-9]*[.][0-9]+)([Ee][+-
]?[0-9]+)?
string          [a-zA-Z_$][0-9a-zA-Z_$./\-]*([\[][0-9]+([:][0-9]+)?[\]])?
estring           [0-9a-zA-Z_$./\?\-\+\*\%]+([\[][0-9]+([:][0-9]+)?[\]])?
nstring         "\\\""([0-9]+[a-zA-Z_][0-9a-zA-Z_]*)|([a-zA-Z_][0-9a-zA-
Z_]*)"\\\""
qstring           ["]([^\n"]|"\\\n"|"\\\"")+["]

%%

<SKIP>"\\\n"                    { echo; libc_lineno++; }
<SKIP>[^\n;]*                   { echo; RN(SDF_SKIP); }
<SKIP>[;]               { echo; PREV_STATE; RN(';'); }
<SKIP>[\n]              { echo; PREV_STATE; libc_lineno++; }
```

```
{ws}                        { echo; }
[\n]|"\\\n"                 { echo; libc_lineno++; }

"/*"                        { echo; NEXT_STATE PS; }
<PS>"*"+"/"                         { echo; PREV_STATE; }

<PS>[^*\n]*                 { echo; }
<PS>"*"+[^*/\n]*            { echo; }

["]                         { echo; RN(libctext[0]); }

{qstring}                   { char *ch;
                              if (yy_start != (1 + 2 * QSTR)) {
                                REJECT;
                              }
                              for (ch=libctext;(*ch)!='\0';ch++) {
                                if ((*ch) == '\n')
                                  libc_lineno++;
                              }
                              libclval.string = copy_string(libctext);
                              echo; PREV_STATE; RN(L_QSTRING);
                            }

<INITIAL>"library"                   { echo; NEXT_STATE LIB; NEXT_STATE NAME;
RN(K_LIBRARY); }
<LIB>[;]                       { echo; RN(libctext[0]); }
<LIB>"aux_no_pulldown_pin_property" { echo; RN(K_ANPPP); }
<LIB>"bus_naming_style"              { echo; NEXT_STATE QSTR; RN(K_BNS); }
<LIB>"comment"                       { echo; NEXT_STATE QSTR; RN(K_COMMENT); }
<LIB>"current_unit"                  { echo; NEXT_STATE CU; RN(K_CU); }
<LIB>"date"                    { echo; NEXT_STATE QSTR; RN(K_DATE); }
<LIB>"delay_model"                   { echo; NEXT_STATE DM; RN(K_DM); }
<LIB>"in_place_swap_mode"            { echo; NEXT_STATE IPO; RN(K_IPSM); }
<LIB>"leakage_power_unit"            { echo; NEXT_STATE PU; RN(K_LPU); }
<LIB>"max_wired_emitters"            { echo; RN(K_MWE); }
<LIB>"multiple_drivers_legal"        { echo; NEXT_STATE BOOL; RN(K_MDL); }
<LIB>"nom_process"                   { echo; RN(K_NP); }
<LIB>"nom_temperature"               { echo; RN(K_NT); }
<LIB>"nom_voltage"                   { echo; RN(K_NV); }
<LIB>"nonpaired_twin_inc_delay_func"      { echo; NEXT_STATE MMP;
RN(K_NTIDF); }
<LIB>"no_pulldown_pin_property"           { echo; RN(K_NPPP); }
<LIB>"piece_type"              { echo; NEXT_STATE PT; RN(K_PT); }
<LIB>"power_unit"              { echo; NEXT_STATE PU; RN(K_PU); }
<LIB>"preferred_output_pad_slew_rate_control"   { echo; RN(K_POPSRC); }
<LIB>"preferred_output_pad_voltage"       { echo; RN(K_POPV); }
<LIB>"preferred_input_pad_voltage"        { echo; RN(K_PIPV); }
<LIB>"pulling_resistance_unit"            { echo; NEXT_STATE RU; RN(K_PRU); }
<LIB>"reference_capacitance"         { echo; RN(K_RC); }          /* cmos2 */
<LIB>"revision"                      { echo; NEXT_STATE REV; RN(K_REVISION); }
<LIB>"simulation"              { echo; NEXT_STATE BOOL; RN(K_SIMULATION); }
<LIB>"time_unit"               { echo; NEXT_STATE TU; RN(K_TU); }
<LIB>"unconnected_pin_property"           { echo; RN(K_UPP); }
<LIB>"voltage_unit"                  { echo; NEXT_STATE VU; RN(K_VU); }
<LIB>"wired_logic_function"          { echo; NEXT_STATE WLF; RN(K_WLF); }
```

```
<LIB>"default_cell_leakage_power"    { echo; RN(K_DCLP); }
<LIB>"default_cell_power"            { echo; RN(K_DCP); }
<LIB>"default_connection_class"          { echo; RN(K_DCC); }
<LIB>"default_edge_rate_breakpoint_f0"   { echo; RN(K_DF0); }    /* cmos2 */
<LIB>"default_edge_rate_breakpoint_f1"   { echo; RN(K_DF1); }    /* cmos2 */
<LIB>"default_edge_rate_breakpoint_r0"   { echo; RN(K_DR0); }    /* cmos2 */
<LIB>"default_edge_rate_breakpoint_r1"   { echo; RN(K_DR1); }    /* cmos2 */
<LIB>"default_emitter_count"         { echo; RN(K_DEC); }
<LIB>"default_fall_delay_intercept"  { echo; RN(K_DFDI); }
<LIB>"default_fall_nonpaired_twin"   { echo; RN(K_DFNT); }
<LIB>"default_fall_pin_resistance"   { echo; RN(K_DFPR); }
<LIB>"default_fall_wire_resistance"  { echo; RN(K_DFWR); }
<LIB>"default_fall_wor_emmiter"          { echo; RN(K_DFWE); }
<LIB>"default_fall_wor_intercept"    { echo; RN(K_DFWI); }
<LIB>"default_fanout_load"           { echo; RN(K_DFL); }
<LIB>"default_inout_pin_cap"         { echo; RN(K_DIOPC); }
<LIB>"default_inout_pin_fall_res"    { echo; RN(K_DIOFR); }
<LIB>"default_inout_pin_rise_res"    { echo; RN(K_DIORR); }
<LIB>"default_input_pin_cap"         { echo; RN(K_DIPC); }
<LIB>"default_intrinsic_fall"        { echo; RN(K_DIF); }
<LIB>"default_intrinsic_rise"        { echo; RN(K_DIR); }
<LIB>"default_max_capacitance"           { echo; RN(K_DMC); }
<LIB>"default_max_fanout"            { echo; RN(K_DMFO); }
<LIB>"default_max_transition"        { echo; RN(K_DMT); }
<LIB>"default_max_utilization"           { echo; RN(K_DMU); }
<LIB>"default_min_porosity"          { echo; RN(K_DMP); }
<LIB>"default_operating_conditions"  { echo; RN(K_DOC); }
<LIB>"default_output_pin_cap"        { echo; RN(K_DOPC); }
<LIB>"default_output_pin_fall_res"   { echo; RN(K_DOFR); }
<LIB>"default_output_pin_rise_res"   { echo; RN(K_DORR); }
<LIB>"default_pin_limit"             { echo; RN(K_DPL); }
<LIB>"default_pin_power"             { echo; RN(K_DPP); }
<LIB>"default_rc_fall_coefficient"   { echo; RN(K_DRFC); }          /* cmos2 */
<LIB>"default_rc_rise_coefficient"   { echo; RN(K_DRRC); }          /* cmos2 */
<LIB>"default_rise_delay_intercept"  { echo; RN(K_DRDI); }
<LIB>"default_rise_nonpaired_twin"   { echo; RN(K_DRNT); }
<LIB>"default_rise_pin_resistance"   { echo; RN(K_DRPR); }
<LIB>"default_rise_wire_resistance"  { echo; RN(K_DRWR); }
<LIB>"default_rise_wor_emitter"          { echo; RN(K_DRWE); }
<LIB>"default_rise_wor_intercept"    { echo; RN(K_DRWI); }
<LIB>"default_setup_coefficient"     { echo; RN(K_DSC); }           /* cmos2 */
<LIB>"default_slope_fall"            { echo; RN(K_DSF); }
<LIB>"default_slope_rise"            { echo; RN(K_DSR); }
<LIB>"default_wire_load"             { echo; NEXT_STATE NAME; RN(K_DWL); }
<LIB>"default_wire_load_area"        { echo; RN(K_DWLA); }
<LIB>"default_wire_load_capacitance"     { echo; RN(K_DWLC); }
<LIB>"default_wire_load_mode"        { echo; NEXT_STATE WLM; RN(K_DWLM); }
<LIB>"default_wire_load_resistance"  { echo; RN(K_DWLR); }
<LIB>"default_wire_load_selection"   { echo; NEXT_STATE NAME; RN(K_DWLS); }

<LIB>"k_process_cell_rise"           { echo; scaling_attr = 0; RN(P_CR); }
<LIB>"k_process_cell_fall"           { echo; scaling_attr = 0; RN(P_CF); }
<LIB>"k_process_cell_leakage_power"  { echo; scaling_attr = 0; RN(P_CLP); }
<LIB>"k_process_cell_power"          { echo; scaling_attr = 0; RN(P_CP); }
<LIB>"k_process_drive_current"           { echo; scaling_attr = 0; RN(P_DC);
}
```

```
<LIB>"k_process_drive_fall"         { echo; scaling_attr = 0; RN(P_DF); }
      /* old */
<LIB>"k_process_drive_rise"         { echo; scaling_attr = 0; RN(P_DR); }
      /* old */
<LIB>"k_process_fall_delay_intercept"     { echo; scaling_attr = 0;
RN(P_FDI); }
<LIB>"k_process_fall_pin_resistance"      { echo; scaling_attr = 0;
RN(P_FPR); }
<LIB>"k_process_fall_propagation"   { echo; scaling_attr = 0; RN(P_FP); }
<LIB>"k_process_fall_transition"    { echo; scaling_attr = 0; RN(P_FT); }
<LIB>"k_process_fall_wire_resistance"     { echo; scaling_attr = 0;
RN(P_FWR); }
<LIB>"k_process_fall_wor_emitter"   { echo; scaling_attr = 0; RN(P_FWE); }
<LIB>"k_process_fall_wor_intercept" { echo; scaling_attr = 0; RN(P_FWI); }
<LIB>"k_process_hold_fall"          { echo; scaling_attr = 0; RN(P_HF); }
<LIB>"k_process_hold_rise"          { echo; scaling_attr = 0; RN(P_HR); }
<LIB>"k_process_internal_power"           { echo; scaling_attr = 0; RN(P_IP);
}
<LIB>"k_process_intrinsic_fall"           { echo; scaling_attr = 0; RN(P_IF);
}
<LIB>"k_process_intrinsic_rise"           { echo; scaling_attr = 0; RN(P_IR);
}
<LIB>"k_process_min_period"         { echo; scaling_attr = 0; RN(P_MP); }
<LIB>"k_process_min_pulse_width_high"     { echo; scaling_attr = 0;
RN(P_MPWH); }
<LIB>"k_process_min_pulse_width_low"      { echo; scaling_attr = 0;
RN(P_MPWL); }
<LIB>"k_process_nochange_fall"            { echo; scaling_attr = 0; RN(P_NF);
}
<LIB>"k_process_nochange_rise"            { echo; scaling_attr = 0; RN(P_NR);
}
<LIB>"k_process_pin_cap"            { echo; scaling_attr = 0; RN(P_PC); }
<LIB>"k_process_pin_power"          { echo; scaling_attr = 0; RN(P_PP); }
<LIB>"k_process_recovery_fall"            { echo; scaling_attr = 0; RN(P_RF);
}
<LIB>"k_process_recovery_rise"            { echo; scaling_attr = 0; RN(P_RR);
}
<LIB>"k_process_removal_fall"       { echo; scaling_attr = 0; RN(P_REF); }
<LIB>"k_process_removal_rise"       { echo; scaling_attr = 0; RN(P_RER); }
<LIB>"k_process_rise_delay_intercept"     { echo; scaling_attr = 0;
RN(P_RDI); }
<LIB>"k_process_rise_pin_resistance"      { echo; scaling_attr = 0;
RN(P_RPR); }
<LIB>"k_process_rise_propagation"   { echo; scaling_attr = 0; RN(P_RP); }
<LIB>"k_process_rise_transition"    { echo; scaling_attr = 0; RN(P_RT); }
<LIB>"k_process_rise_wire_resistance"     { echo; scaling_attr = 0;
RN(P_RWR); }
<LIB>"k_process_rise_wor_emitter"   { echo; scaling_attr = 0; RN(P_RWE); }
<LIB>"k_process_rise_wor_intercept" { echo; scaling_attr = 0; RN(P_RWI); }
<LIB>"k_process_setup_fall"         { echo; scaling_attr = 0; RN(P_SF); }
<LIB>"k_process_setup_rise"         { echo; scaling_attr = 0; RN(P_SR); }
<LIB>"k_process_skew_fall"          { echo; scaling_attr = 0; RN(P_SKF); }
<LIB>"k_process_skew_rise"          { echo; scaling_attr = 0; RN(P_SKR); }
<LIB>"k_process_slope_fall"         { echo; scaling_attr = 0; RN(P_SLF); }
      /* old */
<LIB>"k_process_slope_rise"         { echo; scaling_attr = 0; RN(P_SLR); }
      /* old */
```

```
<LIB>"k_process_wire_cap"           { echo; scaling_attr = 0; RN(P_WC); }
<LIB>"k_process_wire_res"           { echo; scaling_attr = 0; RN(P_WR); }

<LIB>"k_temp_cell_rise"             { echo; scaling_attr = 1; RN(P_CR); }
<LIB>"k_temp_cell_fall"             { echo; scaling_attr = 1; RN(P_CF); }
<LIB>"k_temp_cell_leakage_power"    { echo; scaling_attr = 1; RN(P_CLP); }
<LIB>"k_temp_cell_power"            { echo; scaling_attr = 1; RN(P_CP); }
<LIB>"k_temp_drive_current"         { echo; scaling_attr = 1; RN(P_DC); }
<LIB>"k_temp_drive_fall"            { echo; scaling_attr = 1; RN(P_DF); }
      /* old */
<LIB>"k_temp_drive_rise"            { echo; scaling_attr = 1; RN(P_DR); }
      /* old */
<LIB>"k_temp_fall_delay_intercept"  { echo; scaling_attr = 1; RN(P_FDI); }
<LIB>"k_temp_fall_pin_resistance"   { echo; scaling_attr = 1; RN(P_FPR); }
<LIB>"k_temp_fall_propagation"            { echo; scaling_attr = 1; RN(P_FP);
}
<LIB>"k_temp_fall_transition"       { echo; scaling_attr = 1; RN(P_FT); }
<LIB>"k_temp_fall_wire_resistance"  { echo; scaling_attr = 1; RN(P_FWR); }
<LIB>"k_temp_fall_wor_emitter"            { echo; scaling_attr = 1;
RN(P_FWE); }
<LIB>"k_temp_fall_wor_intercept"    { echo; scaling_attr = 1; RN(P_FWI); }
<LIB>"k_temp_hold_fall"             { echo; scaling_attr = 1; RN(P_HF); }
<LIB>"k_temp_hold_rise"             { echo; scaling_attr = 1; RN(P_HR); }
<LIB>"k_temp_internal_power"        { echo; scaling_attr = 1; RN(P_IP); }
<LIB>"k_temp_intrinsic_fall"        { echo; scaling_attr = 1; RN(P_IF); }
<LIB>"k_temp_intrinsic_rise"        { echo; scaling_attr = 1; RN(P_IR); }
<LIB>"k_temp_min_period"            { echo; scaling_attr = 1; RN(P_MP); }
<LIB>"k_temp_min_pulse_width_high"  { echo; scaling_attr = 1; RN(P_MPWH); }
<LIB>"k_temp_min_pulse_width_low"   { echo; scaling_attr = 1; RN(P_MPWL); }
<LIB>"k_temp_nochange_fall"         { echo; scaling_attr = 1; RN(P_NF); }
<LIB>"k_temp_nochange_rise"         { echo; scaling_attr = 1; RN(P_NR); }
<LIB>"k_temp_pin_cap"               { echo; scaling_attr = 1; RN(P_PC); }
<LIB>"k_temp_pin_power"             { echo; scaling_attr = 1; RN(P_PP); }
<LIB>"k_temp_recovery_fall"         { echo; scaling_attr = 1; RN(P_RF); }
<LIB>"k_temp_recovery_rise"         { echo; scaling_attr = 1; RN(P_RR); }
<LIB>"k_temp_removal_fall"          { echo; scaling_attr = 1; RN(P_REF); }
<LIB>"k_temp_removal_rise"          { echo; scaling_attr = 1; RN(P_RER); }
<LIB>"k_temp_rise_delay_intercept"  { echo; scaling_attr = 1; RN(P_RDI); }
<LIB>"k_temp_rise_pin_resistance"   { echo; scaling_attr = 1; RN(P_RPR); }
<LIB>"k_temp_rise_propagation"            { echo; scaling_attr = 1; RN(P_RP);
}
<LIB>"k_temp_rise_transition"       { echo; scaling_attr = 1; RN(P_RT); }
<LIB>"k_temp_rise_wire_resistance"  { echo; scaling_attr = 1; RN(P_RWR); }
<LIB>"k_temp_rise_wor_emitter"            { echo; scaling_attr = 1;
RN(P_RWE); }
<LIB>"k_temp_rise_wor_intercept"    { echo; scaling_attr = 1; RN(P_RWI); }
<LIB>"k_temp_setup_fall"            { echo; scaling_attr = 1; RN(P_SF); }
<LIB>"k_temp_setup_rise"            { echo; scaling_attr = 1; RN(P_SR); }
<LIB>"k_temp_skew_fall"             { echo; scaling_attr = 1; RN(P_SKF); }
<LIB>"k_temp_skew_rise"             { echo; scaling_attr = 1; RN(P_SKR); }
<LIB>"k_temp_slope_fall"            { echo; scaling_attr = 1; RN(P_SLF); }
      /* old */
<LIB>"k_temp_slope_rise"            { echo; scaling_attr = 1; RN(P_SLR); }
      /* old */
<LIB>"k_temp_wire_cap"              { echo; scaling_attr = 1; RN(P_WC); }
<LIB>"k_temp_wire_res"              { echo; scaling_attr = 1; RN(P_WR); }
```

```
<LIB>"k_volt_cell_rise"             { echo; scaling_attr = 2; RN(P_CR); }
<LIB>"k_volt_cell_fall"             { echo; scaling_attr = 2; RN(P_CF); }
<LIB>"k_volt_cell_leakage_power"    { echo; scaling_attr = 2; RN(P_CLP); }
<LIB>"k_volt_cell_power"            { echo; scaling_attr = 2; RN(P_CP); }
<LIB>"k_volt_drive_current"         { echo; scaling_attr = 2; RN(P_DC); }
<LIB>"k_volt_drive_fall"            { echo; scaling_attr = 2; RN(P_DF); }
      /* old */
<LIB>"k_volt_drive_rise"            { echo; scaling_attr = 2; RN(P_DR); }
      /* old */
<LIB>"k_volt_fall_delay_intercept"  { echo; scaling_attr = 2; RN(P_FDI); }
<LIB>"k_volt_fall_pin_resistance"   { echo; scaling_attr = 2; RN(P_FPR); }
<LIB>"k_volt_fall_propagation"          { echo; scaling_attr = 2; RN(P_FP);
}
<LIB>"k_volt_fall_transition"       { echo; scaling_attr = 2; RN(P_FT); }
<LIB>"k_volt_fall_wire_resistance"  { echo; scaling_attr = 2; RN(P_FWR); }
<LIB>"k_volt_fall_wor_emitter"          { echo; scaling_attr = 2;
RN(P_FWE); }
<LIB>"k_volt_fall_wor_intercept"    { echo; scaling_attr = 2; RN(P_FWI); }
<LIB>"k_volt_hold_fall"             { echo; scaling_attr = 2; RN(P_HF); }
<LIB>"k_volt_hold_rise"             { echo; scaling_attr = 2; RN(P_HR); }
<LIB>"k_volt_internal_power"        { echo; scaling_attr = 2; RN(P_IP); }
<LIB>"k_volt_intrinsic_fall"        { echo; scaling_attr = 2; RN(P_IF); }
<LIB>"k_volt_intrinsic_rise"        { echo; scaling_attr = 2; RN(P_IR); }
<LIB>"k_volt_min_period"            { echo; scaling_attr = 2; RN(P_MP); }
<LIB>"k_volt_min_pulse_width_high"  { echo; scaling_attr = 2; RN(P_MPWH); }
<LIB>"k_volt_min_pulse_width_low"   { echo; scaling_attr = 2; RN(P_MPWL); }
<LIB>"k_volt_nochange_fall"         { echo; scaling_attr = 2; RN(P_NF); }
<LIB>"k_volt_nochange_rise"         { echo; scaling_attr = 2; RN(P_NR); }
<LIB>"k_volt_pin_cap"               { echo; scaling_attr = 2; RN(P_PC); }
<LIB>"k_volt_pin_power"             { echo; scaling_attr = 2; RN(P_PP); }
<LIB>"k_volt_recovery_fall"         { echo; scaling_attr = 2; RN(P_RF); }
<LIB>"k_volt_recovery_rise"         { echo; scaling_attr = 2; RN(P_RR); }
<LIB>"k_volt_removal_fall"          { echo; scaling_attr = 2; RN(P_REF); }
<LIB>"k_volt_removal_rise"          { echo; scaling_attr = 2; RN(P_RER); }
<LIB>"k_volt_rise_delay_intercept"  { echo; scaling_attr = 2; RN(P_RDI); }
<LIB>"k_volt_rise_pin_resistance"   { echo; scaling_attr = 2; RN(P_RPR); }
<LIB>"k_volt_rise_propagation"          { echo; scaling_attr = 2; RN(P_RP);
}
<LIB>"k_volt_rise_transition"       { echo; scaling_attr = 2; RN(P_RT); }
<LIB>"k_volt_rise_wire_resistance"  { echo; scaling_attr = 2; RN(P_RWR); }
<LIB>"k_volt_rise_wor_emitter"          { echo; scaling_attr = 2;
RN(P_RWE); }
<LIB>"k_volt_rise_wor_intercept"    { echo; scaling_attr = 2; RN(P_RWI); }
<LIB>"k_volt_setup_fall"            { echo; scaling_attr = 2; RN(P_SF); }
<LIB>"k_volt_setup_rise"            { echo; scaling_attr = 2; RN(P_SR); }
<LIB>"k_volt_skew_fall"             { echo; scaling_attr = 2; RN(P_SKF); }
<LIB>"k_volt_skew_rise"             { echo; scaling_attr = 2; RN(P_SKR); }
<LIB>"k_volt_slope_fall"            { echo; scaling_attr = 2; RN(P_SLF); }
      /* old */
<LIB>"k_volt_slope_rise"            { echo; scaling_attr = 2; RN(P_SLR); }
      /* old */
<LIB>"k_volt_wire_cap"              { echo; scaling_attr = 2; RN(P_WC); }
<LIB>"k_volt_wire_res"              { echo; scaling_attr = 2; RN(P_WR); }

<LIB>"capacitive_load_unit"         { echo; NEXT_STATE LU; RN(K_CLU); }
<LIB>"define"                       { echo; NEXT_STATE ANY_S; RN(K_DEFINE); }
<LIB>"define_cell_area"             { echo; NEXT_STATE RT; RN(K_DCA); }
```

```
<LIB>"library_features"             { echo; RN(K_LF); }
<LIB>"piece_define"                 { echo; RN(K_PD); }
<LIB>"routing_layers"               { echo; RN(K_RL); }
<LIB>"technology"             { echo; NEXT_STATE NAME; RN(K_TECH); }

<LIB>"cell"                   { echo; NEXT_STATE CELL; NEXT_STATE NAME;
RN(K_CELL); }
<LIB>"input_voltage"                { echo; NEXT_STATE IV; NEXT_STATE NAME;
RN(K_IV); }
<LIB>"lu_table_template"            { echo; NEXT_STATE LTT; NEXT_STATE NAME;
RN(K_LTT); }
<LIB>"operating_conditions"         { echo; NEXT_STATE OC; NEXT_STATE NAME;
RN(K_OC); }
<LIB>"output_voltage"               { echo; NEXT_STATE OV; NEXT_STATE NAME;
RN(K_OV); }
<LIB>"parameterized_cell"           { echo; NEXT_STATE PC; NEXT_STATE NAME;
RN(K_PCELL); }
<LIB>"power_lut_template"           { echo; NEXT_STATE PLT; NEXT_STATE NAME;
RN(K_PLT); }
<LIB>"power_supply"                 { echo; NEXT_STATE PS; NEXT_STATE NAME;
RN(K_PS); }
<LIB>"rise_transition_degradation"  { echo; NEXT_STATE TD; NEXT_STATE NAME;
RN(K_RTD); }
<LIB>"fall_transition_degradation"  { echo; NEXT_STATE TD; NEXT_STATE NAME;
RN(K_FTD); }
<LIB>"scaled_cell"                  { echo; NEXT_STATE CELL; NEXT_STATE NAME;
RN(K_SC); }
<LIB>"scaling_factors"              { echo; NEXT_STATE LIB; NEXT_STATE NAME;
RN(K_SF); }
<LIB>"timing_range"                 { echo; NEXT_STATE TR; NEXT_STATE NAME;
RN(K_TR); }
<LIB>"type"                   { echo; NEXT_STATE TYPE; NEXT_STATE NAME;
RN(K_TYPE); }
<LIB>"wire_load"              { echo; NEXT_STATE WL; NEXT_STATE NAME;
RN(K_WL); }
<LIB>"wire_load_selection"          { echo; NEXT_STATE WLS; NEXT_STATE NAME;
RN(K_WLS); }
<LIB>"wire_load_table"              { echo; NEXT_STATE WLT; NEXT_STATE NAME;
RN(K_WLT); }

 /* ========================================================================
*/

 /* ---- cell group */
<CELL,PC,CENUM>"area"                       { echo; RN(AREA); }
<CELL,PC,CENUM>"auxiliary_pad_cell"         { echo; NEXT_STATE BOOL; RN(C_APC);
}
<CELL,PC,CENUM>"cell_footprint"                  { echo; NEXT_STATE NAME;
RN(C_CF); }
<CELL,PC,CENUM>"cell_power"                 { echo; RN(C_POWER); }
<CELL,PC,CENUM>"cell_leakage_power"         { echo; RN(C_LP); }
<CELL,PC,CENUM>"contention_condition"            { echo; RN(C_CC); }
<CELL,PC,CENUM>"dont_false"                 { echo; NEXT_STATE SA; RN(C_DF); }
<CELL,PC,CENUM>"dont_touch"                 { echo; NEXT_STATE BOOL; RN(C_DT);
}
<CELL,PC,CENUM>"dont_use"                   { echo; NEXT_STATE BOOL; RN(C_DU);
}
```

```
<CELL,PC,CENUM>"geometry_print"                  { echo; RN(C_GP); }
<CELL,PC,CENUM>"handle_negative_constraint"      { echo; NEXT_STATE BOOL;
RN(C_HNC); }
<CELL>"interface_timing"                  { echo; NEXT_STATE BOOL; RN(C_IT);
} /* old version */
<CELL,PC,CENUM>"map_only"                 { echo; NEXT_STATE BOOL; RN(C_MO);
}
<CELL,PC,CENUM>"pad_cell"                 { echo; NEXT_STATE BOOL; RN(C_PC);
}
<CELL,PC,CENUM>"pad_type"                 { echo; RN(C_PT); }
<CELL,PC,CENUM>"pin_limit"                { echo; RN(C_PL); }
<CELL,PC,CENUM>"preferred"                { echo; NEXT_STATE BOOL; RN(C_P); }
<CELL,PC,CENUM>"scaling_factors"          { echo; RN(C_SF); }
<CELL,PC,CENUM>"scan_group"               { echo; NEXT_STATE QSTR; RN(C_SG);
}
<CELL,PC,CENUM>"single_bit_degenerate"           { echo; RN(C_SBD); }
<CELL,PC,CENUM>"vhdl_name"                { echo; RN(C_VN); }

<CELL,PC,CENUM>"pin_equal"                { echo; RN(C_PE); }
<CELL,PC,CENUM>"pin_opposite"             { echo; RN(C_PO); }
<CELL,PC,CENUM>"rail_connection"          { echo; RN(C_RC); }

<CELL,PC>"bundle"              { echo; NEXT_STATE CBO; RN(C_BUNDLE); }
<CELL,PC,CENUM>"bus"                 { echo; NEXT_STATE BUS; RN(C_BUS); }
<CELL,PC>"internal_power"            { echo; NEXT_STATE CIP; RN(C_IP); }
<CELL,PC>"leakage_power"             { echo; NEXT_STATE CLP; RN(C_LPG); }
<CELL,PC,CTC>"ff"              { echo; NEXT_STATE FF; RN(C_FF); }
<CELL,PC,CTC>"ff_bank"               { echo; NEXT_STATE FF; RN(C_FFB); }
<CELL,PC,CTC>"latch"                 { echo; NEXT_STATE CL; RN(C_LATCH); }
<CELL,PC,CTC>"latch_bank"            { echo; NEXT_STATE CL; RN(C_LB); }
<CELL,PC>"lut"                       { echo; NEXT_STATE LUT; RN(C_LUT); }
      /* for FPGA */
<CELL,PC>"memory"              { echo; NEXT_STATE CM; RN(C_MEM); }
<CELL,PC,BUS>"pin"                   { echo; NEXT_STATE PIN; NEXT_STATE NAME;
RN(C_PIN); }
<CELL,PC>"routing_track"             { echo; NEXT_STATE CRT; RN(C_RT); }
<CELL,PC,CTC>"state"                 { echo; NEXT_STATE OST; RN(C_STATE); }
      /* old ff,latch */
<CELL,PC,CTC>"statetable"            { echo; NEXT_STATE CST; RN(C_ST); }
<CELL,PC,CENUM>"test_cell"           { echo; NEXT_STATE CTC; RN(C_TC); }
<CELL,PC,CENUM,BUS>";"               { echo; RN(libctext[0]); }

 /* ---------------------------------------------------------------------
*/

 /* ---- boundle() in cell */

<CBO>"members"                       { echo; RN(MEMBERS); }
<CBO>[;]                       { echo; RN(libctext[0]); }

 /* ---------------------------------------------------------------------
*/

 /* ---- bus() group in cell */

<BUS>"bus_type"                      { echo; RN(BUS_TYPE); }
<BUS>"memory_read"                   { echo; NEXT_STATE MR; RN(MEM_READ); }
```

```
<BUS>"memory_write"                { echo; NEXT_STATE MW; RN(MEM_WRITE); }

 /* ------------------------------------------------------------------------
*/

 /* ---- cell_enum() group */
<CENUM>"cell_property"             { echo; RN(CE_PROPERTY); }
<CENUM>"default_enum"              { echo; NEXT_STATE BOOL; RN(CE_DE); }
<CENUM>"parameterized_pin"         { echo; NEXT_STATE PP; RN(CE_PP); }
 /* ---- parameterized_pin group */
<PP>"pin_properties"               { echo; RN(PP_PROPERTIES); };
<PP>"disabled"                     { echo; NEXT_STATE BOOL; RN(PP_DISABLE);
}
<PP>[;]                            { echo; RN(libctext[0]); }

 /* ------------------------------------------------------------------------
*/

 /* ---- ff(), ff_bank() group <FF> */
 /* ---- latch(), latch_bank() group <CL> */
 /* ---- memory_write() group (MW) */
 /* ---- state() group (OST) old ff,latch */
<FF,OST,MW>"clocked_on"            { echo; RN(CLOCK_ON); }
<FF,OST>"next_state"               { echo; RN(NEXT_ST); }
<FF,OST,CL>"clear"                 { echo; RN(CLEAR); }
<FF,OST,CL>"preset"                { echo; RN(PRESET); }
<FF,OST,CL>"clear_preset_var1"          { echo; RN(CL_PS_V1); }
<FF,OST,CL>"clear_preset_var2"          { echo; RN(CL_PS_V2); }
<FF,OST>"clocked_on_also"          { echo; RN(ON_ALSO); }
<CL,OST,MW>"enable"                { echo; RN(ENABLE); }
<CL,OST>"enable_on_also"           { echo; RN(ON_ALSO); }
<CL,OST>"data_in"             { echo; RN(DATA_IN); }
<OST>"force_01"                    { echo; RN(FORCE_01); }
<OST>"force_10"                    { echo; RN(FORCE_10); }
<OST>"force_00"                    { echo; RN(FORCE_00); }
<OST>"force_11"                    { echo; RN(FORCE_11); }
<FF,MW,CL,OST>[;]             { echo; RN(libctext[0]); }

 /* ------------------------------------------------------------------------
*/

 /* ---- internal_power(), leakage_power() group */
<CIP>"related_input"               { echo; RN(REL_INP); }
<CIP>"related_inputs"              { echo; RN(REL_INPS); }
<CIP>"related_outputs"             { echo; RN(REL_OUTP); }
<CIP>"values"                      { echo; RN(VALUES); }
<PIP>"equal_or_opposite_output"         { echo; RN(IP_EOOO); }
<PIP>"power_level"                 { echo; RN(IP_PL); }
<PIP>"fall_power"             { echo; NEXT_STATE VS; RN(IP_FP); }
<PIP>"rise_power"             { echo; NEXT_STATE VS; RN(IP_RP); }
<PIP>"power"                       { echo; NEXT_STATE VS; RN(IP_POWER); }
<CLP>"value"                       { echo; RN(VALUE); }
<CIP,CLP,PIP>[;]              { echo; RN(libctext[0]); }

 /* ------------------------------------------------------------------------
*/
```

```
 /* ---- lut() group */
<LUT>"input_pins"              { echo; RN(LUT_IP); }
<LUT>[;]                       { echo; RN(libctext[0]); }

 /* ------------------------------------------------------------------------
*/

 /* ---- memory() group */
<CM>"type"                     { echo; RN(CM_TYPE); }
<CM>"ram"                      { echo; RN(CM_RAM); }
<CM>"rom"                      { echo; RN(CM_ROM); }
<CM>"address_width"                 { echo; RN(CM_ADDR_WIDTH); }
<CM>"word_width"               { echo; RN(CM_WORD_WIDTH); }
<CM>"column_address"                { echo; RN(CM_C_ADDR); }
<CM>"row_address"              { echo; RN(CM_R_ADDR); }
<CM>[;]                             { echo; RN(libctext[0]); }

 /* ------------------------------------------------------------------------
*/

 /* ---- pin() group */

<PIN,CBO,BUS,PP>"capacitance"        { echo; RN(PIN_CAP); }
<PIN,CBO,BUS,PP>"clock"              { echo; NEXT_STATE BOOL; RN(PIN_CLK); }
<PIN,CBO,BUS,PP>"clock_gate_enable_pin"   { echo; NEXT_STATE BOOL;
RN(PIN_CGEP); }
<PIN,CBO,BUS,PP>"connection_class"  { echo; RN(PIN_CC); }
<PIN,CBO,BUS,PP>"direction"          { echo; NEXT_STATE DIR; RN(PIN_DIR); }
<PIN,CBO,BUS,PP>"dont_false"         { echo; NEXT_STATE SA; RN(PIN_DF); }
<PIN,CBO,BUS,PP>"drive_current"           { echo; RN(PIN_DC); }
<PIN,CBO,BUS,PP>"driver_type"        { echo; NEXT_STATE DRT; RN(PIN_DT); }
<PIN,CBO,BUS,PP>"edge_rate_breakpoint_f0" { echo; RN(PIN_ERBF0); }
<PIN,CBO,BUS,PP>"edge_rate_breakpoint_f1" { echo; RN(PIN_ERBF1); }
<PIN,CBO,BUS,PP>"edge_rate_breakpoint_r0" { echo; RN(PIN_ERBR0); }
<PIN,CBO,BUS,PP>"edge_rate_breakpoint_r1" { echo; RN(PIN_ERBR1); }
<PIN,CBO,BUS,PP>"edge_rate_fall"          { echo; RN(PIN_ERF); }
<PIN,CBO,BUS,PP>"edge_rate_rise"          { echo; RN(PIN_ERR); }
<PIN,CBO,BUS,PP>"edge_rate_load_fall"          { echo; RN(PIN_ERLF); }
<PIN,CBO,BUS,PP>"edge_rate_load_rise"          { echo; RN(PIN_ERLR); }
<PIN,CBO,BUS,PP>"emitter_count"           { echo; RN(PIN_EC); }
<PIN,CBO,BUS,PP>"fall_current_slop_after_threshold"   { echo; RN(PIN_FCSAT);
}
<PIN,CBO,BUS,PP>"fall_current_slop_before_threshold"  { echo; RN(PIN_FCSBT);
}
<PIN,CBO,BUS,PP>"fall_time_after_threshold"     { echo; RN(PIN_FTAT); }
<PIN,CBO,BUS,PP>"fall_time_before_threshold"    { echo; RN(PIN_FTBT); }
<PIN,CBO,BUS,PP>"fall_wor_emitter"        { echo; RN(PIN_FWE); }
<PIN,CBO,BUS,PP>"fall_wor_intercept"           { echo; RN(PIN_FWI); }
<PIN,CBO,BUS,PP>"fanout_load"        { echo; RN(PIN_FL); }
<PIN,CBO,BUS,PP>"function"           { echo; RN(PIN_FUNCTION); }
<PIN,CBO,BUS,PP>"hysteresis"         { echo; NEXT_STATE BOOL; RN(PIN_H); }
<PIN,CBO,BUS,PP>"input_map"          { echo; RN(PIN_IM); }
<PIN,CBO,BUS,PP>"input_signal_level"      { echo; RN(PIN_ISL); }
<PIN,CBO,BUS,PP>"input_voltage"           { echo; RN(PIN_IV); }
<PIN,CBO,BUS,PP>"internal_node"           { echo; RN(PIN_IN); }         /*
old */
<PIN,CBO,BUS,PP>"inverted_output"    { echo; NEXT_STATE BOOL; RN(PIN_IO); }
```

```
<PIN,CBO,BUS,PP>"is_pad"            { echo; NEXT_STATE BOOL; RN(PIN_IP); }
<PIN,CBO,BUS,PP>"max_fanout"        { echo; RN(PIN_MAX_FO); }
<PIN,CBO,BUS,PP>"max_transition"    { echo; RN(PIN_MAX_TRANS); }
<PIN,CBO,BUS,PP>"max_capacitance"   { echo; RN(PIN_MAX_CAP); }
<PIN,CBO,BUS,PP>"min_fanout"        { echo; RN(PIN_MIN_FO); }
<PIN,CBO,BUS,PP>"min_transition"    { echo; RN(PIN_MIN_TRANS); }
<PIN,CBO,BUS,PP>"min_capacitance"   { echo; RN(PIN_MIN_CAP); }
<PIN,CBO,BUS,PP>"min_period"        { echo; RN(PIN_MP); }
<PIN,CBO,BUS,PP>"min_pulse_width_high"    { echo; RN(PIN_MPWH); }
<PIN,CBO,BUS,PP>"min_pulse_width_low"     { echo; RN(PIN_MPWL); }
<PIN,CBO,BUS,PP>"multicell_pad_pin" { echo; NEXT_STATE BOOL; RN(PIN_MPP); }
<PIN,CBO,BUS,PP>"multiple_drivers_legal"  { echo; NEXT_STATE BOOL;
RN(PIN_MDL); }
<PIN,CBO,BUS,PP>"nextstate_type"    { echo; NEXT_STATE NST; RN(PIN_NST); }
<PIN,CBO,BUS,PP>"output_signal_level"     { echo; RN(PIN_OSL); }
<PIN,CBO,BUS,PP>"output_voltage"    { echo; RN(PIN_OV); }
<PIN,CBO,BUS,PP>"pin_func_type"           { echo; NEXT_STATE PFT;
RN(PIN_PFT); }
<PIN,CBO,BUS,PP>"pin_power"         { echo; RN(PIN_PP); }
<PIN,CBO,BUS,PP>"prefer_tied"       { echo; RN(PIN_PT); }
<PIN,CBO,BUS,PP>"primary_output"    { echo; NEXT_STATE BOOL; RN(PIN_PO); }
<PIN,CBO,BUS,PP>"pulling_current"   { echo; RN(PIN_PC); }
<PIN,CBO,BUS,PP>"pulling_resistance"      { echo; RN(PIN_PR); }
<PIN,CBO,BUS,PP>"reference_capacitance"   { echo; RN(PIN_RC); }
<PIN,CBO,BUS,PP>"rise_current_slop_after_threshold"   { echo; RN(PIN_RCSAT);
}
<PIN,CBO,BUS,PP>"rise_current_slop_before_threshold"  { echo; RN(PIN_RCSBT);
}
<PIN,CBO,BUS,PP>"rise_time_after_threshold"     { echo; RN(PIN_RTAT); }
<PIN,CBO,BUS,PP>"rise_time_before_threshold"    { echo; RN(PIN_RTBT); }
<PIN,CBO,BUS,PP>"rise_wor_emitter"        { echo; RN(PIN_RWE); }
<PIN,CBO,BUS,PP>"rise_wor_intercept"            { echo; RN(PIN_RWI); }
<PIN,CBO,BUS,PP>"slew_control"                  { echo; NEXT_STATE PSC;
RN(PIN_SC); }
<PIN,CBO,BUS,PP>"state_function"          { echo; RN(PIN_SF); }
<PIN,CBO,BUS,PP>"three_state"             { echo; RN(PIN_TS); }
<PIN,CBO,BUS,PP>"vhdl_name"               { echo; RN(PIN_VN); }
<PIN,CBO,BUS,PP>"wire_capacitance"        { echo; RN(PIN_WC); }
<PIN,CBO,BUS,PP>"wired_connection_class"  { echo; RN(PIN_WCC); }
<PIN,CBO,BUS,PP>"x_function"              { echo; RN(PIN_XF); }

<PIN,CBO,BUS>"timing"                     { echo; NEXT_STATE TI; RN(TIMING);
}
<PIN,CBO,BUS>"min_pulse_width"                  { echo; NEXT_STATE MPW;
RN(MIN_PLUSE_WIDTH); }
<PIN,CBO,BUS>"minimum_period"             { echo; NEXT_STATE MP;
RN(MIN_PERIOD); }
<PIN,CBO,BUS>"internal_power"             { echo; NEXT_STATE PIP;
RN(PIN_IPO); }
<PIN>[;]                            { echo; RN(libctext[0]); }

 /* ---------------------------------------------------------------------
*/

 /* ---- memory_read(), memory_write() */
<MR,MW>"address"                { echo; RN(ADDRESS); }
<MR>[;]                               { echo; RN(libctext[0]); }
```

```
 /* ------------------------------------------------------------------------
*/

 /* ---- routing_track() */
<CRT>"tracks"                          { echo; RN(TRACKS); }
<CRT>"total_track_area"                { echo; RN(TRACK_AREA); }
<CRT>[;]                         { echo; RN(libctext[0]); }

 /* ------------------------------------------------------------------------
*/

 /* ---- statetable() */
<CST>"table"                     { echo; NEXT_STATE QSTR; RN(TABLE); }
<CST>[;]                   { echo; RN(libctext[0]); }

 /* ------------------------------------------------------------------------
*/

 /* ---- test_cell() */
<CTC>"pin"                       { echo; NEXT_STATE NAME; RN(CTC_PIN); }
<CTC>"direction"                 { echo; NEXT_STATE DIR; RN(CTC_DIR); }
<CTC>"function"                        { echo; RN(CTC_FUNC); }
<PIN,CBO,BUS,PP,CTC>"signal_type"    { echo; NEXT_STATE SIG_T; RN(CTC_ST); }
<PIN,CBO,BUS,PP,CTC>"test_output_only"     { echo; NEXT_STATE BOOL;
RN(CTC_TOO); }
<CTC>[;]                         { echo; RN(libctext[0]); }

 /* ========================================================================
*/

 /* ---- timing() group */
<TI>"edge_rate_sensitivity_f0"             { echo; RN(TI_ERSF0); }
<TI>"edge_rate_sensitivity_f1"             { echo; RN(TI_ERSF1); }
<TI>"edge_rate_sensitivity_r0"             { echo; RN(TI_ERSR0); }
<TI>"edge_rate_sensitivity_r1"             { echo; RN(TI_ERSR1); }
<TI>"fall_resistance"                { echo; RN(TI_FR); }
<TI>"rise_resistance"                { echo; RN(TI_RR); }
<TI>"intrinsic_fall"                 { echo; RN(TI_IF); }
<TI>"intrinsic_rise"                 { echo; RN(TI_IR); }
<TI>"related_bus_pins"               { echo; RN(TI_RBP); }
<TI>"related_output_pin"             { echo; NEXT_STATE ROP; RN(TI_ROP); }
<TI,PIP>"related_pin"                { echo; RN(TI_RP); }
<TI,MPW,MP>"sdf_cond"                { echo; /* NEXT_STATE QSTR; */
RN(TI_SDF_C); }
<TI>"sdf_cond_start"                 { echo; /* NEXT_STATE QSTR; */
RN(TI_SDF_CS); }
<TI>"sdf_cond_end"                   { echo; /* NEXT_STATE QSTR; */
RN(TI_SDF_CE); }
<TI>"sdf_edges"                        { echo; NEXT_STATE SDF_E; RN(TI_SDF_E); }
<TI>"slope_fall"                 { echo; RN(TI_SF); }
<TI>"slope_rise"                 { echo; RN(TI_SR); }
<TI>"timing_type"                { echo; NEXT_STATE TIT; RN(TI_TT); }
<TI>"timing_sense"                     { echo; NEXT_STATE TIS; RN(TI_TS); }
<TI,MPW,MP,PIP,CLP>"when"              { echo; RN(TI_WHEN); }
<TI>"when_start"                 { echo; RN(TI_WS); }
<TI>"when_end"                         { echo; RN(TI_WE); }
```

```
<TI>"fall_delay_intercept"          { echo; RN(TI_FDI); }
<TI>"fall_nonpaired_twin"           { echo; RN(TI_FNT); }
<TI>"fall_pin_resistance"           { echo; RN(TI_FPR); }
<TI>"fall_wire_resistance"          { echo; RN(TI_FWR); }
<TI>"rise_delay_intercept"          { echo; RN(TI_RDI); }
<TI>"rise_nonpaired_twin"           { echo; RN(TI_RNT); }
<TI>"rise_pin_resistance"           { echo; RN(TI_RPR); }
<TI>"rise_wire_resistance"          { echo; RN(TI_RWR); }

<TI>"cell_degradation"              { echo; NEXT_STATE VS; RN(CELL_DEGR); }
<TI>"cell_fall"                     { echo; NEXT_STATE VS; RN(CELL_FALL); }
<TI>"cell_rise"                     { echo; NEXT_STATE VS; RN(CELL_RISE); }
<TI>"rise_propagation"              { echo; NEXT_STATE VS; RN(R_PROP); }
<TI>"fall_propagation"              { echo; NEXT_STATE VS; RN(F_PROP); }
<TI>"rise_transition"               { echo; NEXT_STATE VS; RN(R_TRANS); }
<TI>"fall_transition"               { echo; NEXT_STATE VS; RN(F_TRANS); }
<TI>"rise_constraint"               { echo; NEXT_STATE VS; RN(R_CONS); }
<TI>"fall_constraint"               { echo; NEXT_STATE VS; RN(F_CONS); }
<TI>[;]                             { echo; RN(libctext[0]); }

 /* ------------------------------------------------------------------------
*/

 /* ---- min_pulse_width() group */
<MPW>"constraint_high"              { echo; RN(MPW_CH); }
<MPW>"constraint_low"               { echo; RN(MPW_CL); }
<MPW>[;]                      { echo; RN(libctext[0]); }

 /* ------------------------------------------------------------------------
*/

 /* ---- min_pulse_width() group */
<MP>"constraint"                { echo; RN(MP_C); }
<MP>[;]                             { echo; RN(libctext[0]); }

 /* ========================================================================
*/

 /* ---- input_voltage group */
<IV>"vil"                     { echo; RN(IV_L); }
<IV>"vih"                     { echo; RN(IV_H); }
<IV>"vimin"                   { echo; RN(IV_MIN); }
<IV>"vimax"                   { echo; RN(IV_MAX); }
<IV>[;]                             { echo; RN(libctext[0]); }

 /* ---- lu_table_template group */
<LTT,PLT>"variable_1"               { echo; RN(TBL_VAR1); }
<LTT,PLT>"variable_2"               { echo; RN(TBL_VAR2); }
<LTT,PLT>"variable_3"               { echo; RN(TBL_VAR3); }
<LTT,PLT,VS,CIP>"index_1"           { echo; RN(TBL_IDX1); }
<LTT,PLT,VS,CIP>"index_2"           { echo; RN(TBL_IDX2); }
<LTT,PLT,VS,CIP>"index_3"           { echo; RN(TBL_IDX3); }
<LTT,PLT>"input_net_transition"           { echo; RN(LTT_INT); }
<PLT>"input_transition_time"        { echo; RN(LTT_INT); }
<LTT,PLT>"total_output_net_capacitance"   { echo; RN(LTT_TONC); }
<LTT,PLT>"output_net_length"        { echo; RN(LTT_ONL); }
```

```
<LTT,PLT>"output_net_wire_cap"            { echo; RN(LTT_ONWC); }
<LTT,PLT>"output_net_pin_cap"       { echo; RN(LTT_ONPC); }
<LTT,ROP>"related_out_total_output_net_capacitance"        { echo;
RN(LTT_ROTONC); }
<LTT,ROP>"related_out_output_net_length"             { echo; RN(LTT_ROONL);
}
<LTT,ROP>"related_out_output_net_wire_cap"                 { echo;
RN(LTT_ROONWC); }
<LTT,ROP>"related_out_output_net_pin_cap"            { echo; RN(LTT_ROONPC);
}
<LTT>"constrained_pin_transition"   { echo; RN(LTT_CPT); }
<LTT>"related_pin_transition"       { echo; RN(LTT_RPT); }
<LTT,PLT>"output_pin_transition"    { echo; RN(LTT_OPT); }
<LTT>"connect_delay"                { echo; RN(LTT_CD); }
<VS>"values"                        { echo; RN(VALUES); }
<LTT,PLT,VS,ROP>[;]                 { echo; RN(libctext[0]); }

 /* ---- operating_conditions group */
<OC>"process"                       { echo; RN(OC_PROCESS); };
<OC>"temperature"             { echo; RN(OC_TEMP); };
<OC>"tree_type"                     { echo; NEXT_STATE TT; RN(OC_TREE); };
<OC>"voltage"                       { echo; RN(OC_VOLT); };
<OC,PS>"power_rail"                 { echo; RN(OC_PR); };
<OC>[;]                             { echo; RN(libctext[0]); }

 /* ---- output_voltage group */
<OV>"vol"                     { echo; RN(OV_L); }
<OV>"voh"                     { echo; RN(OV_H); }
<OV>"vomin"                   { echo; RN(OV_MIN); }
<OV>"vomax"                   { echo; RN(OV_MAX); }
<OV>[;]                             { echo; RN(libctext[0]); }

 /* ---- parameterized_cell group */
<PC>"cell_enum"                     { echo; NEXT_STATE CENUM; NEXT_STATE
NAME; RN(PC_CELL_ENUM); }

 /* ---- power_lut_template group PLT */

 /* ---- power_supply group */
<PS>"default_power_rail"            { echo; RN(PS_DPR); }

 /* ---- rise_transition_degradation group */
 /* ---- fall_transition_degradation group */
<TD>"values"                        { echo; RN(VALUES); }
<TD>[;]                             { echo; RN(libctext[0]); }

 /* ---- scaled_cell group ?  */
 /* ---- scaling_factors group ? */

 /* ---- timing_range group */
<TR>"faster_factor"                 { echo; RN(TR_FF); }
<TR>"slower_factor"                 { echo; RN(TR_SF); }
<TR>[;]                             { echo; RN(libctext[0]); }

 /* ---- type group */
<TYPE>"base_type"             { echo; RN(TYPE_BASE); }
<TYPE>"array"                       { echo; RN(TYPE_ARRAY); }
```

```
<TYPE>"bit_from"                { echo; RN(TYPE_FROM); }
<TYPE>"bit_to"                      { echo; RN(TYPE_TO); }
<TYPE>"bit_width"               { echo; RN(TYPE_WIDTH); }
<TYPE>"data_type"               { echo; RN(TYPE_DT); }
<TYPE>"bit"                     { echo; RN(TYPE_BIT); }
<TYPE>"downto"                      { echo; NEXT_STATE BOOL; RN(TYPE_DOWNTO);
}
<TYPE>[;]                       { echo; RN(libctext[0]); }

 /* ---- wire_load group */
<WL>"area"                      { echo; RN(AREA); }
<WL>"capacitance"               { echo; RN(PIN_CAP); }
<WL>"resistance"                { echo; RN(WL_RES); }
<WL>"slope"                     { echo; RN(WL_SLOPE); }
<WL,WLT>"fanout_length"             { echo; RN(WL_FL); }
<WL>[;]                             { echo; RN(libctext[0]); }

 /* ---- wire_load_selection group */
<WLS>"wire_load_from_area"          { echo; RN(WLS_WLFA); };
<WLS,NAME,QSTR>[;]                  { echo; RN(libctext[0]); }

 /* ---- wire_load_table group */
<WLT>"fanout_area"                  { echo; RN(WLT_AREA); }
<WLT>"fanout_capacitance"           { echo; RN(WLT_CAP); }
<WLT>"fanout_resistance"            { echo; RN(WLT_RES); }
<WLT>[;]                        { echo; RN(libctext[0]); }

 /* ------------------------------------------------------------------------
*/

 /* ---- boolean value */
<BOOL>"true"                        { echo; PREV_STATE; RN(BOOL_T); }
<BOOL>"false"                       { echo; PREV_STATE; RN(BOOL_F); }
<BOOL>";"                       { echo; PREV_STATE; RN(libctext[0]); }

 /* ---- current unit */
<CU>"A"                             { echo; PREV_STATE; RN(CU_A); }
<CU>"mA"                        { echo; PREV_STATE; RN(CU_MA); }
<CU>"uA"                        { echo; PREV_STATE; RN(CU_UA); }
<CU>";"                             { echo; PREV_STATE; RN(libctext[0]); }

 /* ---- load unit */
<LU>"mf"                        { echo; PREV_STATE; RN(LU_MF); }
<LU>"uf"                        { echo; PREV_STATE; RN(LU_UF); }
<LU>"nf"                        { echo; PREV_STATE; RN(LU_NF); }
<LU>"pf"                        { echo; PREV_STATE; RN(LU_PF); }
<LU>"ff"                        { echo; PREV_STATE; RN(LU_FF); }
<LU>";"                             { echo; PREV_STATE; RN(libctext[0]); }

 /* ---- vote unit */
<VU>"V"                             { echo; PREV_STATE; RN(VU_V); }
<VU>"mV"                        { echo; PREV_STATE; RN(VU_MV); }
<VU>";"                             { echo; PREV_STATE; RN(libctext[0]); }

 /* ---- power unit */
<PU>"W"                             { echo; PREV_STATE; RN(PU_W); }
<PU>"mW"                        { echo; PREV_STATE; RN(PU_MW); }
```

```
<PU>"uW"                        { echo; PREV_STATE; RN(PU_UW); }
<PU>"nW"                        { echo; PREV_STATE; RN(PU_NW); }
<PU>"pW"                        { echo; PREV_STATE; RN(PU_PW); }
<PU>";"                             { echo; PREV_STATE; RN(libctext[0]); }

 /* ---- resistance unit */
<RU>"ohm"                       { echo; PREV_STATE; RN(RU_OHM); }
<RU>"kohm"                      { echo; PREV_STATE; RN(RU_KOHM); }
<RU>";"                             { echo; PREV_STATE; RN(libctext[0]); }

 /* ---- time unit */
<TU>"ns"                        { echo; PREV_STATE; RN(TU_NS); }
<TU>"ps"                        { echo; PREV_STATE; RN(TU_PS); }
<TU>";"                             { echo; PREV_STATE; RN(libctext[0]); }

<DM>"generic_ecl"               { echo; PREV_STATE; RN(DM_G_ECL); }
<DM>"generic_cmos"                  { echo; PREV_STATE; RN(DM_G_CMOS); }
<DM>"table_lookup"                  { echo; PREV_STATE; RN(DM_TBL_LOOKUP); }
<DM>"cmos2"                     { echo; PREV_STATE; RN(DM_CMOS2); }
<DM>"piecewise_cmos"                { echo; PREV_STATE; RN(DM_P_COMS); }
<DM>";"                             { echo; PREV_STATE; RN(libctext[0]); }

<IPO>"match_footprint"              { echo; PREV_STATE; RN(IPO_MF); }
<IPO>"no_swapping"                  { echo; PREV_STATE; RN(IPO_NS); }
<IPO>"ignore_footprint"             { echo; PREV_STATE; RN(IPO_IF); }
<IPO>";"                        { echo; PREV_STATE; RN(libctext[0]); }

<MMP>"max"                      { echo; PREV_STATE; RN(MMP_MAX); }
<MMP>"min"                      { echo; PREV_STATE; RN(MMP_MIN); }
<MMP>"plus"                     { echo; PREV_STATE; RN(MMP_PLUS); }
<MMP>";"                        { echo; PREV_STATE; RN(libctext[0]); }

<PT>"piece_length"                  { echo; PREV_STATE; RN(PT_LENGTH); }
<PT>"piece_wire_cap"                { echo; PREV_STATE; RN(PT_WIRE_CAP); }
<PT>"piece_pin_cap"                 { echo; PREV_STATE; RN(PT_PIN_CAP); }
<PT>"piece_total_cap"               { echo; PREV_STATE; RN(PT_TOTAL_CAP); }
<PT>";"                             { echo; PREV_STATE; RN(libctext[0]); }

 /* ---- wire logic function */
<WLF>"wired_and"                { echo; PREV_STATE; RN(WLF_WAND); }
<WLF>"wired_or"                     { echo; PREV_STATE; RN(WLF_WOR); }
<WLF>";"                        { echo; PREV_STATE; RN(libctext[0]); }

<WLM>"top"                      { echo; PREV_STATE; RN(WLM_T); }
<WLM>"segmented"                { echo; PREV_STATE; RN(WLM_S); }
<WLM>"enclosed"                     { echo; PREV_STATE; RN(WLM_E); }
<WLM>";"                        { echo; PREV_STATE; RN(libctext[0]); }

 /* ---- resource type (pad type) */
<RT>"pad_slots"                     { echo; PREV_STATE; RN(RT_PS); }
<RT>"pad_driver_sites"              { echo; PREV_STATE; RN(RT_PDS); }
<RT>"pad_input_driver_sites"        { echo; PREV_STATE; RN(RT_PIDS); }
<RT>"pad_output_driver_sites"       { echo; PREV_STATE; RN(RT_PODS); }
<RT>[,]                             { echo; RN(libctext[0]); }
<RT>[;]                             { echo; PREV_STATE; RN(libctext[0]); }

 /* ---- any string (include keywords) */
```

```
<ANY_S>{string}                         { echo; libclval.string =
copy_string(libctext); RN(L_STRING); }
<ANY_S>";"                        { echo; PREV_STATE; RN(libctext[0]); }

 /* ---- tree_type */
<TT>"best_case_tree"                    { echo; PREV_STATE; RN(TT_BEST); }
<TT>"balanced_tree"                     { echo; PREV_STATE; RN(TT_BAL); }
<TT>"worst_case_tree"                   { echo; PREV_STATE; RN(TT_WORST); }
<TT>";"                                 { echo; PREV_STATE; RN(libctext[0]); }

 /* ----------------------------------------------------------------------
*/

 /* ---- dont_false condition */
<SA>"sa0"                         { echo; PREV_STATE; RN(SA_SA0); }
<SA>"sa1"                         { echo; PREV_STATE; RN(SA_SA1); }
<SA>"sa01"                        { echo; PREV_STATE; RN(SA_SA01); }
<SA>";"                                 { echo; PREV_STATE; RN(libctext[0]); }

 /* ---- direction */
<DIR>"input"                            { echo; PREV_STATE; RN(DIR_INPUT); }
<DIR>"output"                           { echo; PREV_STATE; RN(DIR_OUTPUT); }
<DIR>"inout"                            { echo; PREV_STATE; RN(DIR_INOUT); }
<DIR>"internal"                         { echo; PREV_STATE; RN(DIR_INTERNAL); }
<DIR>";"                          { echo; PREV_STATE; RN(libctext[0]); }

 /* ---- driver_type */
<DRT>"pull_up"                          { echo; PREV_STATE; RN(DRT_PULL_UP); }
<DRT>"pull_down"                  { echo; PREV_STATE; RN(DRT_PULL_DOWN); }
<DRT>"open_drain"                 { echo; PREV_STATE; RN(DRT_OPEN_DRAIN); }
<DRT>"open_source"                      { echo; PREV_STATE; RN(DRT_OPEN_SOURCE);
}
<DRT>"bus_hold"                         { echo; PREV_STATE; RN(DRT_BUS_HOLD); }
<DRT>"resistive"                  { echo; PREV_STATE; RN(DRT_RES); }
<DRT>"resistive_0"                      { echo; PREV_STATE; RN(DRT_RES0); }
<DRT>"resistive_1"                      { echo; PREV_STATE; RN(DRT_RES1); }
<DRT>";"                          { echo; PREV_STATE; RN(libctext[0]); }

 /* ---- nextstate_type */
<NST>"data"                       { echo; PREV_STATE; RN(N_DATA); }
<NST>"preset"                           { echo; PREV_STATE; RN(N_PRESET); }
<NST>"clear"                            { echo; PREV_STATE; RN(N_CLEAR); }
<NST>"load"                       { echo; PREV_STATE; RN(N_LOAD); }
<NST>"scan_in"                          { echo; PREV_STATE; RN(N_SCAN_IN); }
<NST>"scan_enable"                      { echo; PREV_STATE; RN(N_SCAN_ENABLE); }
<NST>";"                          { echo; PREV_STATE; RN(libctext[0]); }

 /* ---- pin_func_type */
<PFT>"clock_enable"                     { echo; PREV_STATE; RN(CLK_ENABLE); }
<PFT>"active_high"                      { echo; PREV_STATE; RN(ACT_HIGH); }
<PFT>"active_low"                 { echo; PREV_STATE; RN(ACT_LOW); }
<PFT>"active_rising"                    { echo; PREV_STATE; RN(ACT_RISING); }
<PFT>"active_falling"                   { echo; PREV_STATE; RN(ACT_FALLING); }
<PFT>";"                          { echo; PREV_STATE; RN(libctext[0]); }

 /* ---- slew_control */
<PSC>"none"                       { echo; PREV_STATE; RN(NONE_SC); }
```

```
<PSC>"low"                      { echo; PREV_STATE; RN(LOW_SC); }
<PSC>"medium"                       { echo; PREV_STATE; RN(MED_SC); }
<PSC>"high"                     { echo; PREV_STATE; RN(HIGH_SC); }
<PSC>";"                        { echo; PREV_STATE; RN(libctext[0]); }

<SDF_E>"noedge"                     { echo; PREV_STATE; RN(NO_EDGE); }
<SDF_E>"both_edges"                 { echo; PREV_STATE; RN(BOTH_EDGES); }
<SDF_E>"start_edge"                 { echo; PREV_STATE; RN(START_EDGE); }
<SDF_E>"end_edge"               { echo; PREV_STATE; RN(END_EDGE); }
<SDF_E>";"                      { echo; PREV_STATE; RN(libctext[0]); }

 /* ------------------------------------------------------------------
*/

 /* ---- timing_type */
<TIT>"rising_edge"                  { echo; PREV_STATE; RN(TIT_RE); }
<TIT>"falling_edge"                 { echo; PREV_STATE; RN(TIT_FE); }
<TIT>"preset"                       { echo; PREV_STATE; RN(TIT_PS); }
<TIT>"clear"                        { echo; PREV_STATE; RN(TIT_CL); }
<TIT>"hold_rising"                  { echo; PREV_STATE; RN(TIT_HR); }
<TIT>"hold_falling"                 { echo; PREV_STATE; RN(TIT_HF); }
<TIT>"setup_rising"                 { echo; PREV_STATE; RN(TIT_SR); }
<TIT>"setup_falling"                { echo; PREV_STATE; RN(TIT_SF); }
<TIT>"recovery_rising"              { echo; PREV_STATE; RN(TIT_RR); }
<TIT>"recovery_falling"             { echo; PREV_STATE; RN(TIT_RF); }
<TIT>"three_state_disable"          { echo; PREV_STATE; RN(TIT_TSD); }
<TIT>"three_state_enable"           { echo; PREV_STATE; RN(TIT_TSE); }
<TIT>"removal_rising"               { echo; PREV_STATE; RN(TIT_RMR); }
<TIT>"removal_falling"              { echo; PREV_STATE; RN(TIT_RMF); }
<TIT>"combinational"                { echo; PREV_STATE; RN(TIT_C); }

<TIT>"skew_rising"                  { echo; PREV_STATE; RN(TIT_SKR); }
<TIT>"skew_falling"                 { echo; PREV_STATE; RN(TIT_SKF); }
<TIT>"non_seq_hold_rising"          { echo; PREV_STATE; RN(TIT_NSHR); }
<TIT>"non_seq_hold_falling"         { echo; PREV_STATE; RN(TIT_NSHF); }
<TIT>"non_seq_setup_rising"         { echo; PREV_STATE; RN(TIT_NSSR); }
<TIT>"non_seq_setup_falling"        { echo; PREV_STATE; RN(TIT_NSSF); }
<TIT>"nochange_high_high"           { echo; PREV_STATE; RN(TIT_NCHH); }
<TIT>"nochange_high_low"            { echo; PREV_STATE; RN(TIT_NCHL); }
<TIT>"nochange_low_high"            { echo; PREV_STATE; RN(TIT_NCLH); }
<TIT>"nochange_low_low"             { echo; PREV_STATE; RN(TIT_NCLL); }
<TIT>";"                        { echo; PREV_STATE; RN(libctext[0]); }

 /* ---- timing_sense */
<TIS>"positive_unate"               { echo; PREV_STATE; RN(TIS_POS); }
<TIS>"negative_unate"               { echo; PREV_STATE; RN(TIS_NEG); }
<TIS>"non_unate"                { echo; PREV_STATE; RN(TIS_NON); }
<TIS>";"                        { echo; PREV_STATE; RN(libctext[0]); }

 /* ---- signal_type in test_cell */
<SIG_T>"test_scan_in"               { echo; PREV_STATE; RN(ST_TSI); }
<SIG_T>"test_scan_in_inverted"          { echo; PREV_STATE; RN(ST_TSII); }
<SIG_T>"test_scan_out"              { echo; PREV_STATE; RN(ST_TSO); }
<SIG_T>"test_scan_out_inverted"         { echo; PREV_STATE; RN(ST_TSOI); }
<SIG_T>"test_scan_enable"           { echo; PREV_STATE; RN(ST_TSE); }
<SIG_T>"test_scan_enable_inverted"  { echo; PREV_STATE; RN(ST_TSEI); }
<SIG_T>"test_scan_clock"            { echo; PREV_STATE; RN(ST_TSC); }
```

```
<SIG_T>"test_scan_clock_a"          { echo; PREV_STATE; RN(ST_TSCA); }
<SIG_T>"test_scan_clock_b"          { echo; PREV_STATE; RN(ST_TSCB); }
<SIG_T>"test_clock"                 { echo; PREV_STATE; RN(ST_TCLK); }
<SIG_T>";"                      { echo; PREV_STATE; RN(libctext[0]); }

<REV>([0-9.-/]+[a-zA-Z]?)+          { echo; PREV_STATE; libclval.string =
copy_string(libctext); RN(REV_V); }
<REV>{string}                       { echo; PREV_STATE; libclval.string =
copy_string(libctext); RN(REV_V); }
<REV>{qstring}                      { echo; PREV_STATE; libclval.string =
copy_string(libctext); RN(REV_V); }
<REV>";"                        { echo; PREV_STATE; RN(libctext[0]); }

 /* ------------------------------------------------------------------------
*/

{estring}               { if (yy_start != (1+ 2*NAME)) {
                            REJECT;
                          }
                          else {
                            echo;
                            libclval.string = copy_string(libctext);
                            RN(L_ESTRING);
                          }
                        }
{integer}               { echo;
                          if (yy_start == (1+ 2*NAME)) {
                            libclval.string = copy_string(libctext);
RN(L_ESTRING);
                          }
                          else {
                            libclval.int_val = atoi(libctext); RN(L_INT);
                          }
                        }
{real}                      { echo; libclval.real_val = atof(libctext);
RN(L_REAL); }
{string}                { enum value_type vt;
                          echo;
                          libclval.string = copy_string(libctext);
                          if (libc_def_find(copy_string(libctext),&vt)) {
                            if (vt == REAL_VT)        { RN(L_DEF_REAL); }
                            else if (vt == BOOL_VT)   { NEXT_STATE BOOL;
RN(L_DEF_BOOL); }
                            else                      { NEXT_STATE QSTR;
RN(L_DEF_TEXT); }
                          }
                          else                        { RN(L_STRING); }
                        }
{nstring}               { echo; libclval.string = copy_string(libctext);
RN(L_NSTRING); }

[:,{}\+\-\*/'!&\|\(\)^=]    { echo; RN(libctext[0]); }

<<EOF>>                     { yyterminate(); }

.                       { echo; lex_error(libctext); }
```

```
%%
/* ================================================ */

void lex_next_QSTR_state(void)
{ NEXT_STATE QSTR; }

/* ------------------------------------------------ */

void lex_next_NAME_state(void)
{ NEXT_STATE NAME; }

/* ------------------------------------------------ */
void lex_next_SKIP_state(void)
{ NEXT_STATE SKIP; }

/* ------------------------------------------------ */

void lex_prev_state()
{ PREV_STATE;
  /* printf(" =====> yystart == %d\n",yy_start); */
}

/* ================================================ */

void lex_error(
      char *S)
{
  fprintf(stderr,"%s(%d): ERROR: lex error near
%s\n",file_name,libc_lineno,S);
  libc_error_count++;
  if (libc_error_count > 10)
    exit(44);
}

/* ================================================ */

void libcerror(
        char *S)
{
  fprintf(stderr,"%s(%d): ERROR: near %s,
%s\n",file_name,libc_lineno,libctext,S);
  libc_error_count++;
  if (libc_error_count > 10)
    exit(44);
}

/* ------------------------------------------------ */

void libc_error(
        char *S)
{
  fprintf(stderr,"%s(%d): ERROR: %s\n",file_name,libc_lineno,S);
  libc_error_count++;
  if (libc_error_count > 10)
    exit(44);
}
```

```
/* ================================================ */
```

```
/*================================================


 ================================================ */

#define LIBC_MEM
#include "libc_def.h"

/* ================================================ */

double atof(char *);

/* ================================================ */

public
int libc_main(
      int argc,
      char *argv[])
{ int i, j, code = 0;
  char *s;
  char tmpFileName[256];       /* for xxx */
  char clfFileName[256];       /* for xxx */
  char tmpFileName1[256];      /* for yyy */
  char clfFileName1[256];      /* for yyy */
  char tmpFileName2[256];      /* for zzz */
  char clfFileName2[256];      /* for zzz */
  char inFileName[256] = "";
  int linear2TLU = 0;
  FILE *xxx_clf = NULL,*yyy_clf,*yyy_con=NULL,*zzz_clf;

  char conFileNameTmp[256];    /* for yyy version 320 */
  char conFileName[256];       /* for yyy version 320 */

  libc_version = 330;

  for (i=1;i<argc;i++) {
    if(argv[i][0] != '-')
      sprintf(inFileName,"%s",argv[i]) ;
    else {
      if(strcmp(argv[i],"-linear2TLU") == 0)
      linear2TLU = 1;
      else if(strcmp(argv[i],"-3.2") == 0)
      libc_version = 320;
      else if(strcmp(&argv[i][1],"o") == 0) {
      char *str = NULL;

        i++;
      for (str=argv[i];(*str)!='\0';str++) {
        if ((*str) == '/')
          break;
        }
        if ((*str) != '\0') {
          sprintf(tmpFileName, "%s_time.tmp",argv[i]);
          sprintf(tmpFileName1,"%s_logic.tmp",argv[i]);
          sprintf(tmpFileName2,"%s_power.tmp",argv[i]);
```

```
}
  else {
    sprintf(tmpFileName, "%s_time..tmp",argv[i]);
    sprintf(tmpFileName1,"%s_logic..tmp",argv[i]);
    sprintf(tmpFileName2,"%s_power..tmp",argv[i]);
}
  sprintf(clfFileName, "%s.time",argv[i]);
  sprintf(clfFileName1,"%s.logic",argv[i]);
  sprintf(clfFileName2,"%s.power",argv[i]);
  if( (xxx_clf = fopen(tmpFileName, "w")) == NULL) {
  fprintf(stderr,"Could not open output file %s.\n",argv[i]);
  return(44);
}
  if( (yyy_clf = fopen(tmpFileName1, "w")) == NULL) {
  fprintf(stderr,"Could not open output file %s.\n",argv[i]);
  return(44);
}
  if( (zzz_clf = fopen(tmpFileName2, "w")) == NULL) {
  fprintf(stderr,"Could not open output file %s.\n",argv[i]);
  return(44);
}
}
else if(strcmp(&argv[i][1],"temp") == 0) {
i++;
libc_t_max = atof(argv[i]); /*max*/
i++;
libc_t_nom = atof(argv[i]); /* nom */
i++;
libc_t_min = atof(argv[i]); /* min */
}
else if(strcmp(&argv[i][1],"volt") == 0) {
i++;  /* WARNING: Notice reverse index convention! */
libc_v_max = atof(argv[i]); /* max */
i++;
libc_v_nom = atof(argv[i]); /* nom */
i++;
libc_v_min = atof(argv[i]); /* min */
}
else if(strcmp(&argv[i][1],"process") == 0) {
i++;
libc_p_max = atof(argv[i]);
i++;
libc_p_nom = atof(argv[i]);
i++;
libc_p_min = atof(argv[i]);
}
else if(strcmp(&argv[i][1],"facts") == 0) {
i++;
cap_scale = atof(argv[i]);
i++;
resist_scale = atof(argv[i]);
i++;
time_scale = atof(argv[i]);
}
else if(strcmp(&argv[i][1],"power_facts") == 0) {
i++;
watt_scale = atof(argv[i]);
```

```
      i++;
      joule_scale = atof(argv[i]);
      }
    }
  }

  if (libc_version < 330 ) {
    sprintf(conFileNameTmp,"%s_con",tmpFileName1);
    sprintf(conFileName,   "%s_con",clfFileName1);
    if( (yyy_con = fopen(conFileNameTmp, "w")) == NULL) {
      fprintf(stderr,"Could not open output file %s.\n",conFileNameTmp);
      return(44);
    }
  }

  if( (!xxx_clf) || (inFileName[0] == '\0') || (argc <= 1)) {
    printf("Usage: %s -o output_file\n",argv[0]);
    printf("            [-3.2]\n");
    printf("            [-temp max nom min] [-process max nom min] [-volt max
nom min]\n");
    printf("            [-facts cap_scale resist_scale time_scale]\n");
    printf("            [-power_facts watt_scale joule_scale]\n");
    printf("            input_file\n");
    return(44);
  }

  libc_error_count = 0;
  fprintf(stderr,"Parsing %s ...\n",inFileName);
  if( code = parse_a_tlib_file(inFileName) || libc_error_count) {
    fprintf(stderr,"Error found, translation failed.\n");
    code = 1;
  }

  if (!code) {
    if (tech_lib->delay_model == UNKNOW_M)
      tech_lib->delay_model = GENERIC_CMOS;
    libc_gen_clf_main(xxx_clf,yyy_clf,yyy_con,zzz_clf,tech_lib);
  }

  fclose(xxx_clf);
  fclose(yyy_clf);
  fclose(yyy_con);
  fclose(zzz_clf);

  if (linear2TLU)
    /* syLinear2TLU(tmpFileName, clfFileName) */ ;
  else {
    if (rename(tmpFileName, clfFileName) == -1) {
      fprintf(stderr,"Cannot create CLF file %s\n", clfFileName);
      return(45);
    }
  }
  if (rename(tmpFileName1, clfFileName1) == -1) {
    fprintf(stderr,"Cannot create CLF file %s\n", clfFileName1);
    return(45);
  }
  if (libc_version < 330) {
```

```
    if (rename(conFileNameTmp, conFileName) == -1) {
      fprintf(stderr,"Cannot create CON file %s\n",conFileName);
      return(45);
    }
  }
  if (rename(tmpFileName2, clfFileName2) == -1) {
    fprintf(stderr,"Cannot create CLF file %s\n", clfFileName2);
    return(45);
  }

  if (code) {
    unlink(clfFileName);
    unlink(clfFileName1);
    unlink(clfFileName2);
    return(44);
  }
  return(0);
}

/* ================================================= */
```

```
#include <stdio.h>
#include <assert.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include "libc_mem.h"
#if DMP_MEM_ANY
static int libc_mem_curr_size=0;
static int libc_mem_peak_size=0;
static int any_unit_rec_curr_msize=0;
static int any_unit_rec_peak_msize=0;
static int any_unit_rec_lcurr_msize=0;
static int any_unit_rec_lpeak_msize=0;
static int libc_name_list_rec_curr_msize=0;
static int libc_name_list_rec_peak_msize=0;
static int libc_name_list_rec_lcurr_msize=0;
static int libc_name_list_rec_lpeak_msize=0;
static int libc_name_list_list_curr_msize=0;
static int libc_name_list_list_peak_msize=0;
static int libc_name_list_list_lcurr_msize=0;
static int libc_name_list_list_lpeak_msize=0;
static int libc_float_list_rec_curr_msize=0;
static int libc_float_list_rec_peak_msize=0;
static int libc_float_list_rec_lcurr_msize=0;
static int libc_float_list_rec_lpeak_msize=0;
static int libc_float_list_list_curr_msize=0;
static int libc_float_list_list_peak_msize=0;
static int libc_float_list_list_lcurr_msize=0;
static int libc_float_list_list_lpeak_msize=0;
static int libc_define_rec_curr_msize=0;
static int libc_define_rec_peak_msize=0;
static int libc_define_rec_lcurr_msize=0;
static int libc_define_rec_lpeak_msize=0;
static int libc_cell_area_rec_curr_msize=0;
static int libc_cell_area_rec_peak_msize=0;
static int libc_cell_area_rec_lcurr_msize=0;
static int libc_cell_area_rec_lpeak_msize=0;
static int libc_k_factor_rec_curr_msize=0;
static int libc_k_factor_rec_peak_msize=0;
static int libc_k_factor_rec_lcurr_msize=0;
static int libc_k_factor_rec_lpeak_msize=0;
static int libc_lib_rec_curr_msize=0;
static int libc_lib_rec_peak_msize=0;
static int libc_lib_rec_lcurr_msize=0;
static int libc_lib_rec_lpeak_msize=0;
static int libc_lu_table_template_rec_curr_msize=0;
static int libc_lu_table_template_rec_peak_msize=0;
static int libc_lu_table_template_rec_lcurr_msize=0;
static int libc_lu_table_template_rec_lpeak_msize=0;
static int libc_oc_power_rail_rec_curr_msize=0;
static int libc_oc_power_rail_rec_peak_msize=0;
static int libc_oc_power_rail_rec_lcurr_msize=0;
static int libc_oc_power_rail_rec_lpeak_msize=0;
static int libc_operating_condition_rec_curr_msize=0;
static int libc_operating_condition_rec_peak_msize=0;
static int libc_operating_condition_rec_lcurr_msize=0;
static int libc_operating_condition_rec_lpeak_msize=0;
```

```
static int libc_power_supply_rec_curr_msize=0;
static int libc_power_supply_rec_peak_msize=0;
static int libc_power_supply_rec_lcurr_msize=0;
static int libc_power_supply_rec_lpeak_msize=0;
static int libc_timing_range_rec_curr_msize=0;
static int libc_timing_range_rec_peak_msize=0;
static int libc_timing_range_rec_lcurr_msize=0;
static int libc_timing_range_rec_lpeak_msize=0;
static int libc_type_rec_curr_msize=0;
static int libc_type_rec_peak_msize=0;
static int libc_type_rec_lcurr_msize=0;
static int libc_type_rec_lpeak_msize=0;
static int libc_fanout_length_rec_curr_msize=0;
static int libc_fanout_length_rec_peak_msize=0;
static int libc_fanout_length_rec_lcurr_msize=0;
static int libc_fanout_length_rec_lpeak_msize=0;
static int libc_wire_load_rec_curr_msize=0;
static int libc_wire_load_rec_peak_msize=0;
static int libc_wire_load_rec_lcurr_msize=0;
static int libc_wire_load_rec_lpeak_msize=0;
static int libc_wire_load_from_area_rec_curr_msize=0;
static int libc_wire_load_from_area_rec_peak_msize=0;
static int libc_wire_load_from_area_rec_lcurr_msize=0;
static int libc_wire_load_from_area_rec_lpeak_msize=0;
static int libc_wire_load_selection_rec_curr_msize=0;
static int libc_wire_load_selection_rec_peak_msize=0;
static int libc_wire_load_selection_rec_lcurr_msize=0;
static int libc_wire_load_selection_rec_lpeak_msize=0;
static int libc_cell_rec_curr_msize=0;
static int libc_cell_rec_peak_msize=0;
static int libc_cell_rec_lcurr_msize=0;
static int libc_cell_rec_lpeak_msize=0;
static int libc_memory_write_rec_curr_msize=0;
static int libc_memory_write_rec_peak_msize=0;
static int libc_memory_write_rec_lcurr_msize=0;
static int libc_memory_write_rec_lpeak_msize=0;
static int libc_bool_opr_rec_curr_msize=0;
static int libc_bool_opr_rec_peak_msize=0;
static int libc_bool_opr_rec_lcurr_msize=0;
static int libc_bool_opr_rec_lpeak_msize=0;
static int libc_pin_rec_curr_msize=0;
static int libc_pin_rec_peak_msize=0;
static int libc_pin_rec_lcurr_msize=0;
static int libc_pin_rec_lpeak_msize=0;
static int libc_ff_latch_rec_curr_msize=0;
static int libc_ff_latch_rec_peak_msize=0;
static int libc_ff_latch_rec_lcurr_msize=0;
static int libc_ff_latch_rec_lpeak_msize=0;
static int libc_internal_power_curr_msize=0;
static int libc_internal_power_peak_msize=0;
static int libc_internal_power_lcurr_msize=0;
static int libc_internal_power_lpeak_msize=0;
static int libc_leakage_power_curr_msize=0;
static int libc_leakage_power_peak_msize=0;
static int libc_leakage_power_lcurr_msize=0;
static int libc_leakage_power_lpeak_msize=0;
static int libc_memory_rec_curr_msize=0;
```

```
static int libc_memory_rec_peak_msize=0;
static int libc_memory_rec_lcurr_msize=0;
static int libc_memory_rec_lpeak_msize=0;
static int libc_piece_value_rec_curr_msize=0;
static int libc_piece_value_rec_peak_msize=0;
static int libc_piece_value_rec_lcurr_msize=0;
static int libc_piece_value_rec_lpeak_msize=0;
static int libc_float_rec_curr_msize=0;
static int libc_float_rec_peak_msize=0;
static int libc_float_rec_lcurr_msize=0;
static int libc_float_rec_lpeak_msize=0;
static int libc_table_val_rec_curr_msize=0;
static int libc_table_val_rec_peak_msize=0;
static int libc_table_val_rec_lcurr_msize=0;
static int libc_table_val_rec_lpeak_msize=0;
static int libc_timing_rec_curr_msize=0;
static int libc_timing_rec_peak_msize=0;
static int libc_timing_rec_lcurr_msize=0;
static int libc_timing_rec_lpeak_msize=0;
static int libc_min_pulse_width_rec_curr_msize=0;
static int libc_min_pulse_width_rec_peak_msize=0;
static int libc_min_pulse_width_rec_lcurr_msize=0;
static int libc_min_pulse_width_rec_lpeak_msize=0;
static int libc_minimum_period_rec_curr_msize=0;
static int libc_minimum_period_rec_peak_msize=0;
static int libc_minimum_period_rec_lcurr_msize=0;
static int libc_minimum_period_rec_lpeak_msize=0;
static int libc_routing_track_rec_curr_msize=0;
static int libc_routing_track_rec_peak_msize=0;
static int libc_routing_track_rec_lcurr_msize=0;
static int libc_routing_track_rec_lpeak_msize=0;
static int libc_def_entry_rec_curr_msize=0;
static int libc_def_entry_rec_peak_msize=0;
static int libc_def_entry_rec_lcurr_msize=0;
static int libc_def_entry_rec_lpeak_msize=0;
static int libc_define_value_rec_curr_msize=0;
static int libc_define_value_rec_peak_msize=0;
static int libc_define_value_rec_lcurr_msize=0;
static int libc_define_value_rec_lpeak_msize=0;
static int libc_glb_const_rec_curr_msize=0;
static int libc_glb_const_rec_peak_msize=0;
static int libc_glb_const_rec_lcurr_msize=0;
static int libc_glb_const_rec_lpeak_msize=0;
static int libc_def_table_rec_curr_msize=0;
static int libc_def_table_rec_peak_msize=0;
static int libc_def_table_rec_lcurr_msize=0;
static int libc_def_table_rec_lpeak_msize=0;
void reset_libc_mem_size(){
  any_unit_rec_lcurr_msize=any_unit_rec_lpeak_msize=0;
  libc_name_list_rec_lcurr_msize=libc_name_list_rec_lpeak_msize=0;
  libc_name_list_list_lcurr_msize=libc_name_list_list_lpeak_msize=0;
  libc_float_list_rec_lcurr_msize=libc_float_list_rec_lpeak_msize=0;
  libc_float_list_list_lcurr_msize=libc_float_list_list_lpeak_msize=0;
  libc_define_rec_lcurr_msize=libc_define_rec_lpeak_msize=0;
  libc_cell_area_rec_lcurr_msize=libc_cell_area_rec_lpeak_msize=0;
  libc_k_factor_rec_lcurr_msize=libc_k_factor_rec_lpeak_msize=0;
  libc_lib_rec_lcurr_msize=libc_lib_rec_lpeak_msize=0;
```

```
libc_lu_table_template_rec_lcurr_msize=libc_lu_table_template_rec_lpeak_msize
=0;
  libc_oc_power_rail_rec_lcurr_msize=libc_oc_power_rail_rec_lpeak_msize=0;

libc_operating_condition_rec_lcurr_msize=libc_operating_condition_rec_lpeak_m
size=0;
  libc_power_supply_rec_lcurr_msize=libc_power_supply_rec_lpeak_msize=0;
  libc_timing_range_rec_lcurr_msize=libc_timing_range_rec_lpeak_msize=0;
  libc_type_rec_lcurr_msize=libc_type_rec_lpeak_msize=0;
  libc_fanout_length_rec_lcurr_msize=libc_fanout_length_rec_lpeak_msize=0;
  libc_wire_load_rec_lcurr_msize=libc_wire_load_rec_lpeak_msize=0;

libc_wire_load_from_area_rec_lcurr_msize=libc_wire_load_from_area_rec_lpeak_m
size=0;

libc_wire_load_selection_rec_lcurr_msize=libc_wire_load_selection_rec_lpeak_m
size=0;
  libc_cell_rec_lcurr_msize=libc_cell_rec_lpeak_msize=0;
  libc_memory_write_rec_lcurr_msize=libc_memory_write_rec_lpeak_msize=0;
  libc_bool_opr_rec_lcurr_msize=libc_bool_opr_rec_lpeak_msize=0;
  libc_pin_rec_lcurr_msize=libc_pin_rec_lpeak_msize=0;
  libc_ff_latch_rec_lcurr_msize=libc_ff_latch_rec_lpeak_msize=0;
  libc_internal_power_lcurr_msize=libc_internal_power_lpeak_msize=0;
  libc_leakage_power_lcurr_msize=libc_leakage_power_lpeak_msize=0;
  libc_memory_rec_lcurr_msize=libc_memory_rec_lpeak_msize=0;
  libc_piece_value_rec_lcurr_msize=libc_piece_value_rec_lpeak_msize=0;
  libc_float_rec_lcurr_msize=libc_float_rec_lpeak_msize=0;
  libc_table_val_rec_lcurr_msize=libc_table_val_rec_lpeak_msize=0;
  libc_timing_rec_lcurr_msize=libc_timing_rec_lpeak_msize=0;

libc_min_pulse_width_rec_lcurr_msize=libc_min_pulse_width_rec_lpeak_msize=0;
  libc_minimum_period_rec_lcurr_msize=libc_minimum_period_rec_lpeak_msize=0;
  libc_routing_track_rec_lcurr_msize=libc_routing_track_rec_lpeak_msize=0;
  libc_def_entry_rec_lcurr_msize=libc_def_entry_rec_lpeak_msize=0;
  libc_define_value_rec_lcurr_msize=libc_define_value_rec_lpeak_msize=0;
  libc_glb_const_rec_lcurr_msize=libc_glb_const_rec_lpeak_msize=0;
  libc_def_table_rec_lcurr_msize=libc_def_table_rec_lpeak_msize=0;
}
void rpt_libc_mem_lsize(int peak){
dmp_clr_statistic(100);

dmp_ins_statistic("libc_mem","any_unit_rec",peak?any_unit_rec_lcurr_msize:any
_unit_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_name_list_rec",peak?libc_name_list_rec_lcu
rr_msize:libc_name_list_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_name_list_list",peak?libc_name_list_list_l
curr_msize:libc_name_list_list_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_float_list_rec",peak?libc_float_list_rec_l
curr_msize:libc_float_list_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_float_list_list",peak?libc_float_list_list
_lcurr_msize:libc_float_list_list_lpeak_msize);
```

```
dmp_ins_statistic("libc_mem","libc_define_rec",peak?libc_define_rec_lcurr_msi
ze:libc_define_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_cell_area_rec",peak?libc_cell_area_rec_lcu
rr_msize:libc_cell_area_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_k_factor_rec",peak?libc_k_factor_rec_lcurr
_msize:libc_k_factor_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_lib_rec",peak?libc_lib_rec_lcurr_msize:lib
c_lib_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_lu_table_template_rec",peak?libc_lu_table_
template_rec_lcurr_msize:libc_lu_table_template_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_oc_power_rail_rec",peak?libc_oc_power_rail
_rec_lcurr_msize:libc_oc_power_rail_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_operating_condition_rec",peak?libc_operati
ng_condition_rec_lcurr_msize:libc_operating_condition_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_power_supply_rec",peak?libc_power_supply_r
ec_lcurr_msize:libc_power_supply_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_timing_range_rec",peak?libc_timing_range_r
ec_lcurr_msize:libc_timing_range_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_type_rec",peak?libc_type_rec_lcurr_msize:l
ibc_type_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_fanout_length_rec",peak?libc_fanout_length
_rec_lcurr_msize:libc_fanout_length_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_wire_load_rec",peak?libc_wire_load_rec_lcu
rr_msize:libc_wire_load_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_wire_load_from_area_rec",peak?libc_wire_lo
ad_from_area_rec_lcurr_msize:libc_wire_load_from_area_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_wire_load_selection_rec",peak?libc_wire_lo
ad_selection_rec_lcurr_msize:libc_wire_load_selection_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_cell_rec",peak?libc_cell_rec_lcurr_msize:l
ibc_cell_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_memory_write_rec",peak?libc_memory_write_r
ec_lcurr_msize:libc_memory_write_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_bool_opr_rec",peak?libc_bool_opr_rec_lcurr
_msize:libc_bool_opr_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_pin_rec",peak?libc_pin_rec_lcurr_msize:lib
c_pin_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_ff_latch_rec",peak?libc_ff_latch_rec_lcurr
_msize:libc_ff_latch_rec_lpeak_msize);
```

```
dmp_ins_statistic("libc_mem","libc_internal_power",peak?libc_internal_power_l
curr_msize:libc_internal_power_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_leakage_power",peak?libc_leakage_power_lcu
rr_msize:libc_leakage_power_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_memory_rec",peak?libc_memory_rec_lcurr_msi
ze:libc_memory_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_piece_value_rec",peak?libc_piece_value_rec
_lcurr_msize:libc_piece_value_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_float_rec",peak?libc_float_rec_lcurr_msize
:libc_float_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_table_val_rec",peak?libc_table_val_rec_lcu
rr_msize:libc_table_val_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_timing_rec",peak?libc_timing_rec_lcurr_msi
ze:libc_timing_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_min_pulse_width_rec",peak?libc_min_pulse_w
idth_rec_lcurr_msize:libc_min_pulse_width_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_minimum_period_rec",peak?libc_minimum_peri
od_rec_lcurr_msize:libc_minimum_period_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_routing_track_rec",peak?libc_routing_track
_rec_lcurr_msize:libc_routing_track_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_def_entry_rec",peak?libc_def_entry_rec_lcu
rr_msize:libc_def_entry_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_define_value_rec",peak?libc_define_value_r
ec_lcurr_msize:libc_define_value_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_glb_const_rec",peak?libc_glb_const_rec_lcu
rr_msize:libc_glb_const_rec_lpeak_msize);

dmp_ins_statistic("libc_mem","libc_def_table_rec",peak?libc_def_table_rec_lcu
rr_msize:libc_def_table_rec_lpeak_msize);
dmp_rpt_statistic(0);
}
void acc_libc_mem_size(int peak){

dmp_ins_statistic("libc_mem","any_unit_rec",peak?any_unit_rec_peak_msize:any_
unit_rec_curr_msize);

dmp_ins_statistic("libc_mem","libc_name_list_rec",peak?libc_name_list_rec_pea
k_msize:libc_name_list_rec_curr_msize);

dmp_ins_statistic("libc_mem","libc_name_list_list",peak?libc_name_list_list_p
eak_msize:libc_name_list_list_curr_msize);

dmp_ins_statistic("libc_mem","libc_float_list_rec",peak?libc_float_list_rec_p
eak_msize:libc_float_list_rec_curr_msize);
```

```
dmp_ins_statistic("libc_mem","libc_float_list_list",peak?libc_float_list_list
_peak_msize:libc_float_list_list_curr_msize);

dmp_ins_statistic("libc_mem","libc_define_rec",peak?libc_define_rec_peak_msiz
e:libc_define_rec_curr_msize);

dmp_ins_statistic("libc_mem","libc_cell_area_rec",peak?libc_cell_area_rec_pea
k_msize:libc_cell_area_rec_curr_msize);

dmp_ins_statistic("libc_mem","libc_k_factor_rec",peak?libc_k_factor_rec_peak_
msize:libc_k_factor_rec_curr_msize);

dmp_ins_statistic("libc_mem","libc_lib_rec",peak?libc_lib_rec_peak_msize:libc
_lib_rec_curr_msize);

dmp_ins_statistic("libc_mem","libc_lu_table_template_rec",peak?libc_lu_table_
template_rec_peak_msize:libc_lu_table_template_rec_curr_msize);

dmp_ins_statistic("libc_mem","libc_oc_power_rail_rec",peak?libc_oc_power_rail
_rec_peak_msize:libc_oc_power_rail_rec_curr_msize);

dmp_ins_statistic("libc_mem","libc_operating_condition_rec",peak?libc_operati
ng_condition_rec_peak_msize:libc_operating_condition_rec_curr_msize);

dmp_ins_statistic("libc_mem","libc_power_supply_rec",peak?libc_power_supply_r
ec_peak_msize:libc_power_supply_rec_curr_msize);

dmp_ins_statistic("libc_mem","libc_timing_range_rec",peak?libc_timing_range_r
ec_peak_msize:libc_timing_range_rec_curr_msize);

dmp_ins_statistic("libc_mem","libc_type_rec",peak?libc_type_rec_peak_msize:li
bc_type_rec_curr_msize);

dmp_ins_statistic("libc_mem","libc_fanout_length_rec",peak?libc_fanout_length
_rec_peak_msize:libc_fanout_length_rec_curr_msize);

dmp_ins_statistic("libc_mem","libc_wire_load_rec",peak?libc_wire_load_rec_pea
k_msize:libc_wire_load_rec_curr_msize);

dmp_ins_statistic("libc_mem","libc_wire_load_from_area_rec",peak?libc_wire_lo
ad_from_area_rec_peak_msize:libc_wire_load_from_area_rec_curr_msize);

dmp_ins_statistic("libc_mem","libc_wire_load_selection_rec",peak?libc_wire_lo
ad_selection_rec_peak_msize:libc_wire_load_selection_rec_curr_msize);

dmp_ins_statistic("libc_mem","libc_cell_rec",peak?libc_cell_rec_peak_msize:li
bc_cell_rec_curr_msize);

dmp_ins_statistic("libc_mem","libc_memory_write_rec",peak?libc_memory_write_r
ec_peak_msize:libc_memory_write_rec_curr_msize);

dmp_ins_statistic("libc_mem","libc_bool_opr_rec",peak?libc_bool_opr_rec_peak_
msize:libc_bool_opr_rec_curr_msize);

dmp_ins_statistic("libc_mem","libc_pin_rec",peak?libc_pin_rec_peak_msize:libc
_pin_rec_curr_msize);
```

```
dmp_ins_statistic("libc_mem","libc_ff_latch_rec",peak?libc_ff_latch_rec_peak_
msize:libc_ff_latch_rec_curr_msize);

dmp_ins_statistic("libc_mem","libc_internal_power",peak?libc_internal_power_p
eak_msize:libc_internal_power_curr_msize);

dmp_ins_statistic("libc_mem","libc_leakage_power",peak?libc_leakage_power_pea
k_msize:libc_leakage_power_curr_msize);

dmp_ins_statistic("libc_mem","libc_memory_rec",peak?libc_memory_rec_peak_msiz
e:libc_memory_rec_curr_msize);

dmp_ins_statistic("libc_mem","libc_piece_value_rec",peak?libc_piece_value_rec
_peak_msize:libc_piece_value_rec_curr_msize);

dmp_ins_statistic("libc_mem","libc_float_rec",peak?libc_float_rec_peak_msize:
libc_float_rec_curr_msize);

dmp_ins_statistic("libc_mem","libc_table_val_rec",peak?libc_table_val_rec_pea
k_msize:libc_table_val_rec_curr_msize);

dmp_ins_statistic("libc_mem","libc_timing_rec",peak?libc_timing_rec_peak_msiz
e:libc_timing_rec_curr_msize);

dmp_ins_statistic("libc_mem","libc_min_pulse_width_rec",peak?libc_min_pulse_w
idth_rec_peak_msize:libc_min_pulse_width_rec_curr_msize);

dmp_ins_statistic("libc_mem","libc_minimum_period_rec",peak?libc_minimum_peri
od_rec_peak_msize:libc_minimum_period_rec_curr_msize);

dmp_ins_statistic("libc_mem","libc_routing_track_rec",peak?libc_routing_track
_rec_peak_msize:libc_routing_track_rec_curr_msize);

dmp_ins_statistic("libc_mem","libc_def_entry_rec",peak?libc_def_entry_rec_pea
k_msize:libc_def_entry_rec_curr_msize);

dmp_ins_statistic("libc_mem","libc_define_value_rec",peak?libc_define_value_r
ec_peak_msize:libc_define_value_rec_curr_msize);

dmp_ins_statistic("libc_mem","libc_glb_const_rec",peak?libc_glb_const_rec_pea
k_msize:libc_glb_const_rec_curr_msize);

dmp_ins_statistic("libc_mem","libc_def_table_rec",peak?libc_def_table_rec_pea
k_msize:libc_def_table_rec_curr_msize);
}
#define dmp_libc_mem_size_inc(sz)  \
    libc_mem_curr_size += sz; \
    if (libc_mem_curr_size>libc_mem_peak_size) \
    libc_mem_peak_size = libc_mem_curr_size;
#define dmp_libc_mem_size_dec(sz) libc_mem_curr_size -= sz
#define dmp_any_unit_rec_msize_inc(sz)  \
    any_unit_rec_lcurr_msize += sz; \
    if (any_unit_rec_lcurr_msize>any_unit_rec_lpeak_msize) \
    any_unit_rec_lpeak_msize = any_unit_rec_lcurr_msize;\
    any_unit_rec_curr_msize += sz; \
    if (any_unit_rec_curr_msize>any_unit_rec_peak_msize) \
```

```
    any_unit_rec_peak_msize = any_unit_rec_curr_msize;
#define dmp_any_unit_rec_msize_dec(sz) \
      any_unit_rec_curr_msize -= sz;\
      any_unit_rec_lcurr_msize -= sz
struct any_unit_rec *new_any_unit_rec(void)
{ struct any_unit_rec *p = (struct any_unit_rec *) get_mb_ptr(1);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(8);
  dmp_any_unit_rec_msize_inc(8);
#endif
  return(p);
}

#define dmp_libc_name_list_rec_msize_inc(sz)  \
    libc_name_list_rec_lcurr_msize += sz; \
    if (libc_name_list_rec_lcurr_msize>libc_name_list_rec_lpeak_msize) \
    libc_name_list_rec_lpeak_msize = libc_name_list_rec_lcurr_msize;\
    libc_name_list_rec_curr_msize += sz; \
    if (libc_name_list_rec_curr_msize>libc_name_list_rec_peak_msize) \
    libc_name_list_rec_peak_msize = libc_name_list_rec_curr_msize;
#define dmp_libc_name_list_rec_msize_dec(sz) \
      libc_name_list_rec_curr_msize -= sz;\
      libc_name_list_rec_lcurr_msize -= sz
struct libc_name_list_rec *new_libc_name_list_rec(void)
{ struct libc_name_list_rec *p = (struct libc_name_list_rec *) get_mb_ptr(1);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(8);
  dmp_libc_name_list_rec_msize_inc(8);
#endif
  return(p);
}

#define dmp_libc_name_list_list_msize_inc(sz)  \
    libc_name_list_list_lcurr_msize += sz; \
    if (libc_name_list_list_lcurr_msize>libc_name_list_list_lpeak_msize) \
    libc_name_list_list_lpeak_msize = libc_name_list_list_lcurr_msize;\
    libc_name_list_list_curr_msize += sz; \
    if (libc_name_list_list_curr_msize>libc_name_list_list_peak_msize) \
    libc_name_list_list_peak_msize = libc_name_list_list_curr_msize;
#define dmp_libc_name_list_list_msize_dec(sz) \
      libc_name_list_list_curr_msize -= sz;\
      libc_name_list_list_lcurr_msize -= sz
struct libc_name_list_list *new_libc_name_list_list(void)
{ struct libc_name_list_list *p = (struct libc_name_list_list *)
get_mb_ptr(2);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(12);
  dmp_libc_name_list_list_msize_inc(12);
#endif
  return(p);
}

#define dmp_libc_float_list_rec_msize_inc(sz)  \
    libc_float_list_rec_lcurr_msize += sz; \
    if (libc_float_list_rec_lcurr_msize>libc_float_list_rec_lpeak_msize) \
    libc_float_list_rec_lpeak_msize = libc_float_list_rec_lcurr_msize;\
    libc_float_list_rec_curr_msize += sz; \
```

```
    if (libc_float_list_rec_curr_msize>libc_float_list_rec_peak_msize) \
    libc_float_list_rec_peak_msize = libc_float_list_rec_curr_msize;
#define dmp_libc_float_list_rec_msize_dec(sz) \
      libc_float_list_rec_curr_msize -= sz;\
      libc_float_list_rec_lcurr_msize -= sz
struct libc_float_list_rec *new_libc_float_list_rec(void)
{ struct libc_float_list_rec *p = (struct libc_float_list_rec *)
get_mb_ptr(1);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(8);
  dmp_libc_float_list_rec_msize_inc(8);
#endif
  return(p);
}

#define dmp_libc_float_list_list_msize_inc(sz)  \
    libc_float_list_list_lcurr_msize += sz; \
    if (libc_float_list_list_lcurr_msize>libc_float_list_list_lpeak_msize) \
    libc_float_list_list_lpeak_msize = libc_float_list_list_lcurr_msize;\
    libc_float_list_list_curr_msize += sz; \
    if (libc_float_list_list_curr_msize>libc_float_list_list_peak_msize) \
    libc_float_list_list_peak_msize = libc_float_list_list_curr_msize;
#define dmp_libc_float_list_list_msize_dec(sz) \
      libc_float_list_list_curr_msize -= sz;\
      libc_float_list_list_lcurr_msize -= sz
struct libc_float_list_list *new_libc_float_list_list(void)
{ struct libc_float_list_list *p = (struct libc_float_list_list *)
get_mb_ptr(1);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(8);
  dmp_libc_float_list_list_msize_inc(8);
#endif
  return(p);
}

#define dmp_libc_define_rec_msize_inc(sz)  \
    libc_define_rec_lcurr_msize += sz; \
    if (libc_define_rec_lcurr_msize>libc_define_rec_lpeak_msize) \
    libc_define_rec_lpeak_msize = libc_define_rec_lcurr_msize;\
    libc_define_rec_curr_msize += sz; \
    if (libc_define_rec_curr_msize>libc_define_rec_peak_msize) \
    libc_define_rec_peak_msize = libc_define_rec_curr_msize;
#define dmp_libc_define_rec_msize_dec(sz) \
      libc_define_rec_curr_msize -= sz;\
      libc_define_rec_lcurr_msize -= sz
struct libc_define_rec *new_libc_define_rec(void)
{ struct libc_define_rec *p = (struct libc_define_rec *) get_mb_ptr(3);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(16);
  dmp_libc_define_rec_msize_inc(16);
#endif
  return(p);
}

#define dmp_libc_cell_area_rec_msize_inc(sz)  \
    libc_cell_area_rec_lcurr_msize += sz; \
    if (libc_cell_area_rec_lcurr_msize>libc_cell_area_rec_lpeak_msize) \
```

```
    libc_cell_area_rec_lpeak_msize = libc_cell_area_rec_lcurr_msize;\
    libc_cell_area_rec_curr_msize += sz; \
    if (libc_cell_area_rec_curr_msize>libc_cell_area_rec_peak_msize) \
    libc_cell_area_rec_peak_msize = libc_cell_area_rec_curr_msize;
#define dmp_libc_cell_area_rec_msize_dec(sz) \
     libc_cell_area_rec_curr_msize -= sz;\
     libc_cell_area_rec_lcurr_msize -= sz
struct libc_cell_area_rec *new_libc_cell_area_rec(void)
{ struct libc_cell_area_rec *p = (struct libc_cell_area_rec *) get_mb_ptr(2);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(12);
  dmp_libc_cell_area_rec_msize_inc(12);
#endif
  return(p);
}

#define dmp_libc_k_factor_rec_msize_inc(sz)  \
    libc_k_factor_rec_lcurr_msize += sz; \
    if (libc_k_factor_rec_lcurr_msize>libc_k_factor_rec_lpeak_msize) \
    libc_k_factor_rec_lpeak_msize = libc_k_factor_rec_lcurr_msize;\
    libc_k_factor_rec_curr_msize += sz; \
    if (libc_k_factor_rec_curr_msize>libc_k_factor_rec_peak_msize) \
    libc_k_factor_rec_peak_msize = libc_k_factor_rec_curr_msize;
#define dmp_libc_k_factor_rec_msize_dec(sz) \
     libc_k_factor_rec_curr_msize -= sz;\
     libc_k_factor_rec_lcurr_msize -= sz
struct libc_k_factor_rec *new_libc_k_factor_rec(void)
{ struct libc_k_factor_rec *p = (struct libc_k_factor_rec *) get_mb_ptr(29);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(548);
  dmp_libc_k_factor_rec_msize_inc(548);
#endif
  return(p);
}

#define dmp_libc_lib_rec_msize_inc(sz)  \
    libc_lib_rec_lcurr_msize += sz; \
    if (libc_lib_rec_lcurr_msize>libc_lib_rec_lpeak_msize) \
    libc_lib_rec_lpeak_msize = libc_lib_rec_lcurr_msize;\
    libc_lib_rec_curr_msize += sz; \
    if (libc_lib_rec_curr_msize>libc_lib_rec_peak_msize) \
    libc_lib_rec_peak_msize = libc_lib_rec_curr_msize;
#define dmp_libc_lib_rec_msize_dec(sz) \
     libc_lib_rec_curr_msize -= sz;\
     libc_lib_rec_lcurr_msize -= sz
struct libc_lib_rec *new_libc_lib_rec(void)
{ struct libc_lib_rec *p = (struct libc_lib_rec *) get_mb_ptr(28);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(392);
  dmp_libc_lib_rec_msize_inc(392);
#endif
  return(p);
}

#define dmp_libc_lu_table_template_rec_msize_inc(sz)  \
    libc_lu_table_template_rec_lcurr_msize += sz; \
```

```
    if
(libc_lu_table_template_rec_lcurr_msize>libc_lu_table_template_rec_lpeak_msiz
e) \
    libc_lu_table_template_rec_lpeak_msize =
libc_lu_table_template_rec_lcurr_msize;\
    libc_lu_table_template_rec_curr_msize += sz; \
    if
(libc_lu_table_template_rec_curr_msize>libc_lu_table_template_rec_peak_msize)
\
    libc_lu_table_template_rec_peak_msize =
libc_lu_table_template_rec_curr_msize;
#define dmp_libc_lu_table_template_rec_msize_dec(sz) \
      libc_lu_table_template_rec_curr_msize -= sz;\
      libc_lu_table_template_rec_lcurr_msize -= sz
struct libc_lu_table_template_rec *new_libc_lu_table_template_rec(void)
{ struct libc_lu_table_template_rec *p = (struct libc_lu_table_template_rec
*) get_mb_ptr(7);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(32);
  dmp_libc_lu_table_template_rec_msize_inc(32);
#endif
  return(p);
}

#define dmp_libc_oc_power_rail_rec_msize_inc(sz)  \
    libc_oc_power_rail_rec_lcurr_msize += sz; \
    if
(libc_oc_power_rail_rec_lcurr_msize>libc_oc_power_rail_rec_lpeak_msize) \
    libc_oc_power_rail_rec_lpeak_msize = libc_oc_power_rail_rec_lcurr_msize;\
    libc_oc_power_rail_rec_curr_msize += sz; \
    if (libc_oc_power_rail_rec_curr_msize>libc_oc_power_rail_rec_peak_msize)
\
    libc_oc_power_rail_rec_peak_msize = libc_oc_power_rail_rec_curr_msize;
#define dmp_libc_oc_power_rail_rec_msize_dec(sz) \
      libc_oc_power_rail_rec_curr_msize -= sz;\
      libc_oc_power_rail_rec_lcurr_msize -= sz
struct libc_oc_power_rail_rec *new_libc_oc_power_rail_rec(void)
{ struct libc_oc_power_rail_rec *p = (struct libc_oc_power_rail_rec *)
get_mb_ptr(2);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(12);
  dmp_libc_oc_power_rail_rec_msize_inc(12);
#endif
  return(p);
}

#define dmp_libc_operating_condition_rec_msize_inc(sz)  \
    libc_operating_condition_rec_lcurr_msize += sz; \
    if
(libc_operating_condition_rec_lcurr_msize>libc_operating_condition_rec_lpeak_
msize) \
    libc_operating_condition_rec_lpeak_msize =
libc_operating_condition_rec_lcurr_msize;\
    libc_operating_condition_rec_curr_msize += sz; \
    if
(libc_operating_condition_rec_curr_msize>libc_operating_condition_rec_peak_ms
ize) \
```

```
    libc_operating_condition_rec_peak_msize =
libc_operating_condition_rec_curr_msize;
#define dmp_libc_operating_condition_rec_msize_dec(sz) \
      libc_operating_condition_rec_curr_msize -= sz;\
      libc_operating_condition_rec_lcurr_msize -= sz
struct libc_operating_condition_rec *new_libc_operating_condition_rec(void)
{ struct libc_operating_condition_rec *p = (struct
libc_operating_condition_rec *) get_mb_ptr(6);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(28);
  dmp_libc_operating_condition_rec_msize_inc(28);
#endif
  return(p);
}

#define dmp_libc_power_supply_rec_msize_inc(sz)  \
    libc_power_supply_rec_lcurr_msize += sz; \
    if (libc_power_supply_rec_lcurr_msize>libc_power_supply_rec_lpeak_msize)
\
    libc_power_supply_rec_lpeak_msize = libc_power_supply_rec_lcurr_msize;\
    libc_power_supply_rec_curr_msize += sz; \
    if (libc_power_supply_rec_curr_msize>libc_power_supply_rec_peak_msize) \
    libc_power_supply_rec_peak_msize = libc_power_supply_rec_curr_msize;
#define dmp_libc_power_supply_rec_msize_dec(sz) \
      libc_power_supply_rec_curr_msize -= sz;\
      libc_power_supply_rec_lcurr_msize -= sz
struct libc_power_supply_rec *new_libc_power_supply_rec(void)
{ struct libc_power_supply_rec *p = (struct libc_power_supply_rec *)
get_mb_ptr(3);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(16);
  dmp_libc_power_supply_rec_msize_inc(16);
#endif
  return(p);
}

#define dmp_libc_timing_range_rec_msize_inc(sz)  \
    libc_timing_range_rec_lcurr_msize += sz; \
    if (libc_timing_range_rec_lcurr_msize>libc_timing_range_rec_lpeak_msize)
\
    libc_timing_range_rec_lpeak_msize = libc_timing_range_rec_lcurr_msize;\
    libc_timing_range_rec_curr_msize += sz; \
    if (libc_timing_range_rec_curr_msize>libc_timing_range_rec_peak_msize) \
    libc_timing_range_rec_peak_msize = libc_timing_range_rec_curr_msize;
#define dmp_libc_timing_range_rec_msize_dec(sz) \
      libc_timing_range_rec_curr_msize -= sz;\
      libc_timing_range_rec_lcurr_msize -= sz
struct libc_timing_range_rec *new_libc_timing_range_rec(void)
{ struct libc_timing_range_rec *p = (struct libc_timing_range_rec *)
get_mb_ptr(3);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(16);
  dmp_libc_timing_range_rec_msize_inc(16);
#endif
  return(p);
}
```

```
#define dmp_libc_type_rec_msize_inc(sz)  \
    libc_type_rec_lcurr_msize += sz; \
    if (libc_type_rec_lcurr_msize>libc_type_rec_lpeak_msize) \
    libc_type_rec_lpeak_msize = libc_type_rec_lcurr_msize;\
    libc_type_rec_curr_msize += sz; \
    if (libc_type_rec_curr_msize>libc_type_rec_peak_msize) \
    libc_type_rec_peak_msize = libc_type_rec_curr_msize;
#define dmp_libc_type_rec_msize_dec(sz) \
      libc_type_rec_curr_msize -= sz;\
      libc_type_rec_lcurr_msize -= sz
struct libc_type_rec *new_libc_type_rec(void)
{ struct libc_type_rec *p = (struct libc_type_rec *) get_mb_ptr(5);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(24);
  dmp_libc_type_rec_msize_inc(24);
#endif
  return(p);
}

#define dmp_libc_fanout_length_rec_msize_inc(sz)  \
    libc_fanout_length_rec_lcurr_msize += sz; \
    if
(libc_fanout_length_rec_lcurr_msize>libc_fanout_length_rec_lpeak_msize) \
    libc_fanout_length_rec_lpeak_msize = libc_fanout_length_rec_lcurr_msize;\
    libc_fanout_length_rec_curr_msize += sz; \
    if (libc_fanout_length_rec_curr_msize>libc_fanout_length_rec_peak_msize)
\
    libc_fanout_length_rec_peak_msize = libc_fanout_length_rec_curr_msize;
#define dmp_libc_fanout_length_rec_msize_dec(sz) \
      libc_fanout_length_rec_curr_msize -= sz;\
      libc_fanout_length_rec_lcurr_msize -= sz
struct libc_fanout_length_rec *new_libc_fanout_length_rec(void)
{ struct libc_fanout_length_rec *p = (struct libc_fanout_length_rec *)
get_mb_ptr(8);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(36);
  dmp_libc_fanout_length_rec_msize_inc(36);
#endif
  return(p);
}

#define dmp_libc_wire_load_rec_msize_inc(sz)  \
    libc_wire_load_rec_lcurr_msize += sz; \
    if (libc_wire_load_rec_lcurr_msize>libc_wire_load_rec_lpeak_msize) \
    libc_wire_load_rec_lpeak_msize = libc_wire_load_rec_lcurr_msize;\
    libc_wire_load_rec_curr_msize += sz; \
    if (libc_wire_load_rec_curr_msize>libc_wire_load_rec_peak_msize) \
    libc_wire_load_rec_peak_msize = libc_wire_load_rec_curr_msize;
#define dmp_libc_wire_load_rec_msize_dec(sz) \
      libc_wire_load_rec_curr_msize -= sz;\
      libc_wire_load_rec_lcurr_msize -= sz
struct libc_wire_load_rec *new_libc_wire_load_rec(void)
{ struct libc_wire_load_rec *p = (struct libc_wire_load_rec *) get_mb_ptr(6);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(28);
  dmp_libc_wire_load_rec_msize_inc(28);
#endif
```

```
  return(p);
}

#define dmp_libc_wire_load_from_area_rec_msize_inc(sz)  \
    libc_wire_load_from_area_rec_lcurr_msize += sz; \
    if
(libc_wire_load_from_area_rec_lcurr_msize>libc_wire_load_from_area_rec_lpeak_
msize) \
    libc_wire_load_from_area_rec_lpeak_msize =
libc_wire_load_from_area_rec_lcurr_msize;\
    libc_wire_load_from_area_rec_curr_msize += sz; \
    if
(libc_wire_load_from_area_rec_curr_msize>libc_wire_load_from_area_rec_peak_ms
ize) \
    libc_wire_load_from_area_rec_peak_msize =
libc_wire_load_from_area_rec_curr_msize;
#define dmp_libc_wire_load_from_area_rec_msize_dec(sz) \
      libc_wire_load_from_area_rec_curr_msize -= sz;\
      libc_wire_load_from_area_rec_lcurr_msize -= sz
struct libc_wire_load_from_area_rec *new_libc_wire_load_from_area_rec(void)
{ struct libc_wire_load_from_area_rec *p = (struct
libc_wire_load_from_area_rec *) get_mb_ptr(3);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(16);
  dmp_libc_wire_load_from_area_rec_msize_inc(16);
#endif
  return(p);
}

#define dmp_libc_wire_load_selection_rec_msize_inc(sz)  \
    libc_wire_load_selection_rec_lcurr_msize += sz; \
    if
(libc_wire_load_selection_rec_lcurr_msize>libc_wire_load_selection_rec_lpeak_
msize) \
    libc_wire_load_selection_rec_lpeak_msize =
libc_wire_load_selection_rec_lcurr_msize;\
    libc_wire_load_selection_rec_curr_msize += sz; \
    if
(libc_wire_load_selection_rec_curr_msize>libc_wire_load_selection_rec_peak_ms
ize) \
    libc_wire_load_selection_rec_peak_msize =
libc_wire_load_selection_rec_curr_msize;
#define dmp_libc_wire_load_selection_rec_msize_dec(sz) \
      libc_wire_load_selection_rec_curr_msize -= sz;\
      libc_wire_load_selection_rec_lcurr_msize -= sz
struct libc_wire_load_selection_rec *new_libc_wire_load_selection_rec(void)
{ struct libc_wire_load_selection_rec *p = (struct
libc_wire_load_selection_rec *) get_mb_ptr(2);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(12);
  dmp_libc_wire_load_selection_rec_msize_inc(12);
#endif
  return(p);
}

#define dmp_libc_cell_rec_msize_inc(sz)  \
    libc_cell_rec_lcurr_msize += sz; \
```

```
    if (libc_cell_rec_lcurr_msize>libc_cell_rec_lpeak_msize) \
    libc_cell_rec_lpeak_msize = libc_cell_rec_lcurr_msize;\
    libc_cell_rec_curr_msize += sz; \
    if (libc_cell_rec_curr_msize>libc_cell_rec_peak_msize) \
    libc_cell_rec_peak_msize = libc_cell_rec_curr_msize;
#define dmp_libc_cell_rec_msize_dec(sz) \
      libc_cell_rec_curr_msize -= sz;\
      libc_cell_rec_lcurr_msize -= sz
struct libc_cell_rec *new_libc_cell_rec(void)
{ struct libc_cell_rec *p = (struct libc_cell_rec *) get_mb_ptr(23);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(132);
  dmp_libc_cell_rec_msize_inc(132);
#endif
  return(p);
}

#define dmp_libc_memory_write_rec_msize_inc(sz)  \
    libc_memory_write_rec_lcurr_msize += sz; \
    if (libc_memory_write_rec_lcurr_msize>libc_memory_write_rec_lpeak_msize)
\
    libc_memory_write_rec_lpeak_msize = libc_memory_write_rec_lcurr_msize;\
    libc_memory_write_rec_curr_msize += sz; \
    if (libc_memory_write_rec_curr_msize>libc_memory_write_rec_peak_msize) \
    libc_memory_write_rec_peak_msize = libc_memory_write_rec_curr_msize;
#define dmp_libc_memory_write_rec_msize_dec(sz) \
      libc_memory_write_rec_curr_msize -= sz;\
      libc_memory_write_rec_lcurr_msize -= sz
struct libc_memory_write_rec *new_libc_memory_write_rec(void)
{ struct libc_memory_write_rec *p = (struct libc_memory_write_rec *)
get_mb_ptr(2);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(12);
  dmp_libc_memory_write_rec_msize_inc(12);
#endif
  return(p);
}

#define dmp_libc_bool_opr_rec_msize_inc(sz)  \
    libc_bool_opr_rec_lcurr_msize += sz; \
    if (libc_bool_opr_rec_lcurr_msize>libc_bool_opr_rec_lpeak_msize) \
    libc_bool_opr_rec_lpeak_msize = libc_bool_opr_rec_lcurr_msize;\
    libc_bool_opr_rec_curr_msize += sz; \
    if (libc_bool_opr_rec_curr_msize>libc_bool_opr_rec_peak_msize) \
    libc_bool_opr_rec_peak_msize = libc_bool_opr_rec_curr_msize;
#define dmp_libc_bool_opr_rec_msize_dec(sz) \
      libc_bool_opr_rec_curr_msize -= sz;\
      libc_bool_opr_rec_lcurr_msize -= sz
struct libc_bool_opr_rec *new_libc_bool_opr_rec(void)
{ struct libc_bool_opr_rec *p = (struct libc_bool_opr_rec *) get_mb_ptr(3);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(16);
  dmp_libc_bool_opr_rec_msize_inc(16);
#endif
  return(p);
}
```

```
#define dmp_libc_pin_rec_msize_inc(sz)  \
    libc_pin_rec_lcurr_msize += sz; \
    if (libc_pin_rec_lcurr_msize>libc_pin_rec_lpeak_msize) \
    libc_pin_rec_lpeak_msize = libc_pin_rec_lcurr_msize;\
    libc_pin_rec_curr_msize += sz; \
    if (libc_pin_rec_curr_msize>libc_pin_rec_peak_msize) \
    libc_pin_rec_peak_msize = libc_pin_rec_curr_msize;
#define dmp_libc_pin_rec_msize_dec(sz) \
      libc_pin_rec_curr_msize -= sz;\
      libc_pin_rec_lcurr_msize -= sz
struct libc_pin_rec *new_libc_pin_rec(void)
{ struct libc_pin_rec *p = (struct libc_pin_rec *) get_mb_ptr(27);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(276);
  dmp_libc_pin_rec_msize_inc(276);
#endif
  return(p);
}

#define dmp_libc_ff_latch_rec_msize_inc(sz)  \
    libc_ff_latch_rec_lcurr_msize += sz; \
    if (libc_ff_latch_rec_lcurr_msize>libc_ff_latch_rec_lpeak_msize) \
    libc_ff_latch_rec_lpeak_msize = libc_ff_latch_rec_lcurr_msize;\
    libc_ff_latch_rec_curr_msize += sz; \
    if (libc_ff_latch_rec_curr_msize>libc_ff_latch_rec_peak_msize) \
    libc_ff_latch_rec_peak_msize = libc_ff_latch_rec_curr_msize;
#define dmp_libc_ff_latch_rec_msize_dec(sz) \
      libc_ff_latch_rec_curr_msize -= sz;\
      libc_ff_latch_rec_lcurr_msize -= sz
struct libc_ff_latch_rec *new_libc_ff_latch_rec(void)
{ struct libc_ff_latch_rec *p = (struct libc_ff_latch_rec *) get_mb_ptr(15);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(64);
  dmp_libc_ff_latch_rec_msize_inc(64);
#endif
  return(p);
}

#define dmp_libc_internal_power_msize_inc(sz)  \
    libc_internal_power_lcurr_msize += sz; \
    if (libc_internal_power_lcurr_msize>libc_internal_power_lpeak_msize) \
    libc_internal_power_lpeak_msize = libc_internal_power_lcurr_msize;\
    libc_internal_power_curr_msize += sz; \
    if (libc_internal_power_curr_msize>libc_internal_power_peak_msize) \
    libc_internal_power_peak_msize = libc_internal_power_curr_msize;
#define dmp_libc_internal_power_msize_dec(sz) \
      libc_internal_power_curr_msize -= sz;\
      libc_internal_power_lcurr_msize -= sz
struct libc_internal_power *new_libc_internal_power(void)
{ struct libc_internal_power *p = (struct libc_internal_power *)
get_mb_ptr(9);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(40);
  dmp_libc_internal_power_msize_inc(40);
#endif
  return(p);
}
```

```
#define dmp_libc_leakage_power_msize_inc(sz)  \
    libc_leakage_power_lcurr_msize += sz; \
    if (libc_leakage_power_lcurr_msize>libc_leakage_power_lpeak_msize) \
    libc_leakage_power_lpeak_msize = libc_leakage_power_lcurr_msize;\
    libc_leakage_power_curr_msize += sz; \
    if (libc_leakage_power_curr_msize>libc_leakage_power_peak_msize) \
    libc_leakage_power_peak_msize = libc_leakage_power_curr_msize;
#define dmp_libc_leakage_power_msize_dec(sz) \
      libc_leakage_power_curr_msize -= sz;\
      libc_leakage_power_lcurr_msize -= sz
struct libc_leakage_power *new_libc_leakage_power(void)
{ struct libc_leakage_power *p = (struct libc_leakage_power *) get_mb_ptr(2);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(12);
  dmp_libc_leakage_power_msize_inc(12);
#endif
  return(p);
}

#define dmp_libc_memory_rec_msize_inc(sz)  \
    libc_memory_rec_lcurr_msize += sz; \
    if (libc_memory_rec_lcurr_msize>libc_memory_rec_lpeak_msize) \
    libc_memory_rec_lpeak_msize = libc_memory_rec_lcurr_msize;\
    libc_memory_rec_curr_msize += sz; \
    if (libc_memory_rec_curr_msize>libc_memory_rec_peak_msize) \
    libc_memory_rec_peak_msize = libc_memory_rec_curr_msize;
#define dmp_libc_memory_rec_msize_dec(sz) \
      libc_memory_rec_curr_msize -= sz;\
      libc_memory_rec_lcurr_msize -= sz
struct libc_memory_rec *new_libc_memory_rec(void)
{ struct libc_memory_rec *p = (struct libc_memory_rec *) get_mb_ptr(2);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(12);
  dmp_libc_memory_rec_msize_inc(12);
#endif
  return(p);
}

#define dmp_libc_piece_value_rec_msize_inc(sz)  \
    libc_piece_value_rec_lcurr_msize += sz; \
    if (libc_piece_value_rec_lcurr_msize>libc_piece_value_rec_lpeak_msize) \
    libc_piece_value_rec_lpeak_msize = libc_piece_value_rec_lcurr_msize;\
    libc_piece_value_rec_curr_msize += sz; \
    if (libc_piece_value_rec_curr_msize>libc_piece_value_rec_peak_msize) \
    libc_piece_value_rec_peak_msize = libc_piece_value_rec_curr_msize;
#define dmp_libc_piece_value_rec_msize_dec(sz) \
      libc_piece_value_rec_curr_msize -= sz;\
      libc_piece_value_rec_lcurr_msize -= sz
struct libc_piece_value_rec *new_libc_piece_value_rec(void)
{ struct libc_piece_value_rec *p = (struct libc_piece_value_rec *)
get_mb_ptr(9);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(40);
  dmp_libc_piece_value_rec_msize_inc(40);
#endif
  return(p);
```

```
}

#define dmp_libc_float_rec_msize_inc(sz)  \
    libc_float_rec_lcurr_msize += sz; \
    if (libc_float_rec_lcurr_msize>libc_float_rec_lpeak_msize) \
    libc_float_rec_lpeak_msize = libc_float_rec_lcurr_msize;\
    libc_float_rec_curr_msize += sz; \
    if (libc_float_rec_curr_msize>libc_float_rec_peak_msize) \
    libc_float_rec_peak_msize = libc_float_rec_curr_msize;
#define dmp_libc_float_rec_msize_dec(sz) \
      libc_float_rec_curr_msize -= sz;\
      libc_float_rec_lcurr_msize -= sz
struct libc_float_rec *new_libc_float_rec(void)
{ struct libc_float_rec *p = (struct libc_float_rec *) get_mb_ptr(0);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(4);
  dmp_libc_float_rec_msize_inc(4);
#endif
  return(p);
}

#define dmp_libc_table_val_rec_msize_inc(sz)  \
    libc_table_val_rec_lcurr_msize += sz; \
    if (libc_table_val_rec_lcurr_msize>libc_table_val_rec_lpeak_msize) \
    libc_table_val_rec_lpeak_msize = libc_table_val_rec_lcurr_msize;\
    libc_table_val_rec_curr_msize += sz; \
    if (libc_table_val_rec_curr_msize>libc_table_val_rec_peak_msize) \
    libc_table_val_rec_peak_msize = libc_table_val_rec_curr_msize;
#define dmp_libc_table_val_rec_msize_dec(sz) \
      libc_table_val_rec_curr_msize -= sz;\
      libc_table_val_rec_lcurr_msize -= sz
struct libc_table_val_rec *new_libc_table_val_rec(void)
{ struct libc_table_val_rec *p = (struct libc_table_val_rec *) get_mb_ptr(5);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(24);
  dmp_libc_table_val_rec_msize_inc(24);
#endif
  return(p);
}

#define dmp_libc_timing_rec_msize_inc(sz)  \
    libc_timing_rec_lcurr_msize += sz; \
    if (libc_timing_rec_lcurr_msize>libc_timing_rec_lpeak_msize) \
    libc_timing_rec_lpeak_msize = libc_timing_rec_lcurr_msize;\
    libc_timing_rec_curr_msize += sz; \
    if (libc_timing_rec_curr_msize>libc_timing_rec_peak_msize) \
    libc_timing_rec_peak_msize = libc_timing_rec_curr_msize;
#define dmp_libc_timing_rec_msize_dec(sz) \
      libc_timing_rec_curr_msize -= sz;\
      libc_timing_rec_lcurr_msize -= sz
struct libc_timing_rec *new_libc_timing_rec(void)
{ struct libc_timing_rec *p = (struct libc_timing_rec *) get_mb_ptr(22);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(120);
  dmp_libc_timing_rec_msize_inc(120);
#endif
  return(p);
```

```
}

#define dmp_libc_min_pulse_width_rec_msize_inc(sz)  \
    libc_min_pulse_width_rec_lcurr_msize += sz; \
    if
(libc_min_pulse_width_rec_lcurr_msize>libc_min_pulse_width_rec_lpeak_msize) \
    libc_min_pulse_width_rec_lpeak_msize =
libc_min_pulse_width_rec_lcurr_msize;\
    libc_min_pulse_width_rec_curr_msize += sz; \
    if
(libc_min_pulse_width_rec_curr_msize>libc_min_pulse_width_rec_peak_msize) \
    libc_min_pulse_width_rec_peak_msize =
libc_min_pulse_width_rec_curr_msize;
#define dmp_libc_min_pulse_width_rec_msize_dec(sz) \
      libc_min_pulse_width_rec_curr_msize -= sz;\
      libc_min_pulse_width_rec_lcurr_msize -= sz
struct libc_min_pulse_width_rec *new_libc_min_pulse_width_rec(void)
{ struct libc_min_pulse_width_rec *p = (struct libc_min_pulse_width_rec *)
get_mb_ptr(3);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(16);
  dmp_libc_min_pulse_width_rec_msize_inc(16);
#endif
  return(p);
}

#define dmp_libc_minimum_period_rec_msize_inc(sz)  \
    libc_minimum_period_rec_lcurr_msize += sz; \
    if
(libc_minimum_period_rec_lcurr_msize>libc_minimum_period_rec_lpeak_msize) \
    libc_minimum_period_rec_lpeak_msize =
libc_minimum_period_rec_lcurr_msize;\
    libc_minimum_period_rec_curr_msize += sz; \
    if
(libc_minimum_period_rec_curr_msize>libc_minimum_period_rec_peak_msize) \
    libc_minimum_period_rec_peak_msize = libc_minimum_period_rec_curr_msize;
#define dmp_libc_minimum_period_rec_msize_dec(sz) \
      libc_minimum_period_rec_curr_msize -= sz;\
      libc_minimum_period_rec_lcurr_msize -= sz
struct libc_minimum_period_rec *new_libc_minimum_period_rec(void)
{ struct libc_minimum_period_rec *p = (struct libc_minimum_period_rec *)
get_mb_ptr(2);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(12);
  dmp_libc_minimum_period_rec_msize_inc(12);
#endif
  return(p);
}

#define dmp_libc_routing_track_rec_msize_inc(sz)  \
    libc_routing_track_rec_lcurr_msize += sz; \
    if
(libc_routing_track_rec_lcurr_msize>libc_routing_track_rec_lpeak_msize) \
    libc_routing_track_rec_lpeak_msize = libc_routing_track_rec_lcurr_msize;\
    libc_routing_track_rec_curr_msize += sz; \
    if (libc_routing_track_rec_curr_msize>libc_routing_track_rec_peak_msize)
\
```

```
    libc_routing_track_rec_peak_msize = libc_routing_track_rec_curr_msize;
#define dmp_libc_routing_track_rec_msize_dec(sz) \
      libc_routing_track_rec_curr_msize -= sz;\
      libc_routing_track_rec_lcurr_msize -= sz
struct libc_routing_track_rec *new_libc_routing_track_rec(void)
{ struct libc_routing_track_rec *p = (struct libc_routing_track_rec *)
get_mb_ptr(3);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(16);
  dmp_libc_routing_track_rec_msize_inc(16);
#endif
  return(p);
}

#define dmp_libc_def_entry_rec_msize_inc(sz)  \
    libc_def_entry_rec_lcurr_msize += sz; \
    if (libc_def_entry_rec_lcurr_msize>libc_def_entry_rec_lpeak_msize) \
    libc_def_entry_rec_lpeak_msize = libc_def_entry_rec_lcurr_msize;\
    libc_def_entry_rec_curr_msize += sz; \
    if (libc_def_entry_rec_curr_msize>libc_def_entry_rec_peak_msize) \
    libc_def_entry_rec_peak_msize = libc_def_entry_rec_curr_msize;
#define dmp_libc_def_entry_rec_msize_dec(sz) \
      libc_def_entry_rec_curr_msize -= sz;\
      libc_def_entry_rec_lcurr_msize -= sz
struct libc_def_entry_rec *new_libc_def_entry_rec(void)
{ struct libc_def_entry_rec *p = (struct libc_def_entry_rec *) get_mb_ptr(2);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(12);
  dmp_libc_def_entry_rec_msize_inc(12);
#endif
  return(p);
}

#define dmp_libc_define_value_rec_msize_inc(sz)  \
    libc_define_value_rec_lcurr_msize += sz; \
    if (libc_define_value_rec_lcurr_msize>libc_define_value_rec_lpeak_msize)
\
    libc_define_value_rec_lpeak_msize = libc_define_value_rec_lcurr_msize;\
    libc_define_value_rec_curr_msize += sz; \
    if (libc_define_value_rec_curr_msize>libc_define_value_rec_peak_msize) \
    libc_define_value_rec_peak_msize = libc_define_value_rec_curr_msize;
#define dmp_libc_define_value_rec_msize_dec(sz) \
      libc_define_value_rec_curr_msize -= sz;\
      libc_define_value_rec_lcurr_msize -= sz
struct libc_define_value_rec *new_libc_define_value_rec(void)
{ struct libc_define_value_rec *p = (struct libc_define_value_rec *)
get_mb_ptr(3);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(16);
  dmp_libc_define_value_rec_msize_inc(16);
#endif
  return(p);
}

#define dmp_libc_glb_const_rec_msize_inc(sz)  \
    libc_glb_const_rec_lcurr_msize += sz; \
    if (libc_glb_const_rec_lcurr_msize>libc_glb_const_rec_lpeak_msize) \
```

```
    libc_glb_const_rec_lpeak_msize = libc_glb_const_rec_lcurr_msize;\
    libc_glb_const_rec_curr_msize += sz; \
    if (libc_glb_const_rec_curr_msize>libc_glb_const_rec_peak_msize) \
    libc_glb_const_rec_peak_msize = libc_glb_const_rec_curr_msize;
#define dmp_libc_glb_const_rec_msize_dec(sz) \
      libc_glb_const_rec_curr_msize -= sz;\
      libc_glb_const_rec_lcurr_msize -= sz
struct libc_glb_const_rec *new_libc_glb_const_rec(void)
{ struct libc_glb_const_rec *p = (struct libc_glb_const_rec *) get_mb_ptr(2);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(12);
  dmp_libc_glb_const_rec_msize_inc(12);
#endif
  return(p);
}

#define dmp_libc_def_table_rec_msize_inc(sz)  \
    libc_def_table_rec_lcurr_msize += sz; \
    if (libc_def_table_rec_lcurr_msize>libc_def_table_rec_lpeak_msize) \
    libc_def_table_rec_lpeak_msize = libc_def_table_rec_lcurr_msize;\
    libc_def_table_rec_curr_msize += sz; \
    if (libc_def_table_rec_curr_msize>libc_def_table_rec_peak_msize) \
    libc_def_table_rec_peak_msize = libc_def_table_rec_curr_msize;
#define dmp_libc_def_table_rec_msize_dec(sz) \
      libc_def_table_rec_curr_msize -= sz;\
      libc_def_table_rec_lcurr_msize -= sz
struct libc_def_table_rec *new_libc_def_table_rec(void)
{ struct libc_def_table_rec *p = (struct libc_def_table_rec *) get_mb_ptr(3);
#if DMP_MEM_ANY
  dmp_libc_mem_size_inc(16);
  dmp_libc_def_table_rec_msize_inc(16);
#endif
  return(p);
}

#endif
/* ========================================================
   routine:  free_any_unit_rec()
   ======================================================== */

void  free_any_unit_rec(struct any_unit_rec *ptr)
{
  if (ptr==NULL) return;
    free_mb_ptr((void *) ptr,1);
#if DMP_MEM_ANY
    dmp_any_unit_rec_msize_dec(8);
    dmp_libc_mem_size_dec(8);
#endif
}

/* ========================================================
   routine: nfree_any_unit_rec()
   ======================================================== */

void nfree_any_unit_rec(struct any_unit_rec **pptr)
{ struct any_unit_rec *ptr= *pptr;
```

```
  if (ptr==NULL) return;
    free_mb_ptr((void *) ptr,1);
#if DMP_MEM_ANY
    dmp_any_unit_rec_msize_dec(8);
    dmp_libc_mem_size_dec(8);
#endif
  *pptr = NULL;
}

/* ========================================================
   routine:  free_libc_name_list_rec()
   ======================================================== */

void  free_libc_name_list_rec(struct libc_name_list_rec *ptr)
{
  struct libc_name_list_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr->name);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,1);
#if DMP_MEM_ANY
    dmp_libc_name_list_rec_msize_dec(8);
    dmp_libc_mem_size_dec(8);
#endif
    ptr = next ;
  }
}

/* ========================================================
   routine: nfree_libc_name_list_rec()
   ======================================================== */

void nfree_libc_name_list_rec(struct libc_name_list_rec **pptr)
{ struct libc_name_list_rec *ptr= *pptr;

  struct libc_name_list_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr->name);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,1);
#if DMP_MEM_ANY
    dmp_libc_name_list_rec_msize_dec(8);
    dmp_libc_mem_size_dec(8);
#endif
    ptr = next ;
  }
  *pptr = NULL;
}

struct libc_name_list_rec * copy_libc_name_list_rec(struct libc_name_list_rec
*sou)
{ struct libc_name_list_rec *tar,*head,*prev;

  if (sou==NULL) return(NULL);
  prev = NULL;
```

```
  head = tar = new_libc_name_list_rec();
  while (sou!=NULL) {
    tar->name       = (char *) copy_string(sou->name);
    sou  = sou->next ;
    if (sou!=NULL)
    { prev = tar;
      tar->next = new_libc_name_list_rec();
      tar = tar->next;
    }
  }
    return(head);
}


/* ========================================================
   routine:  free_libc_name_list_list()
   ======================================================== */

void  free_libc_name_list_list(struct libc_name_list_list *ptr)
{
  struct libc_name_list_list *next;

  while (ptr!=NULL) {
    free_libc_name_list_rec(ptr-> name_list1);
    free_libc_name_list_rec(ptr->name_list2);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,2);
#if DMP_MEM_ANY
    dmp_libc_name_list_list_msize_dec(12);
    dmp_libc_mem_size_dec(12);
#endif
    ptr = next ;
  }
}

/* ========================================================
   routine: nfree_libc_name_list_list()
   ======================================================== */

void nfree_libc_name_list_list(struct libc_name_list_list **pptr)
{ struct libc_name_list_list *ptr= *pptr;

  struct libc_name_list_list *next;

  while (ptr!=NULL) {
    free_libc_name_list_rec(ptr->name_list1);
    free_libc_name_list_rec(ptr-> name_list2);
    next = ptr->next ;
    free_mb_ptr((void *) ptr,2);
#if DMP_MEM_ANY
    dmp_libc_name_list_list_msize_dec(12);
    dmp_libc_mem_size_dec(12);
#endif
    ptr = next ;
  }
  *pptr = NULL;
}
```

```
/* ======================================================
   routine:  free_libc_float_list_rec()
   ====================================================== */

void  free_libc_float_list_rec(struct libc_float_list_rec *ptr)
{
  struct libc_float_list_rec *next;

  while (ptr!=NULL) {
     next = ptr->next ;
    free_mb_ptr((void *) ptr,1);
#if DMP_MEM_ANY
    dmp_libc_float_list_rec_msize_dec(8);
    dmp_libc_mem_size_dec(8);
#endif
    ptr = next ;
  }
}

/* ======================================================
   routine: nfree_libc_float_list_rec()
   ====================================================== */

void nfree_libc_float_list_rec(struct libc_float_list_rec **pptr)
{ struct libc_float_list_rec *ptr= *pptr;

  struct libc_float_list_rec *next;

  while (ptr!=NULL) {
     next = ptr->next ;
    free_mb_ptr((void *) ptr,1);
#if DMP_MEM_ANY
    dmp_libc_float_list_rec_msize_dec(8);
    dmp_libc_mem_size_dec(8);
#endif
    ptr = next ;
  }
  *pptr = NULL;
}

/* ======================================================
   routine:  free_libc_float_list_list()
   ====================================================== */

void  free_libc_float_list_list(struct libc_float_list_list *ptr)
{
  struct libc_float_list_list *next;

  while (ptr!=NULL) {
    free_libc_float_list_rec(ptr->v_list);
    next = ptr->next ;
    free_mb_ptr((void *) ptr,1);
#if DMP_MEM_ANY
    dmp_libc_float_list_list_msize_dec(8);
    dmp_libc_mem_size_dec(8);
#endif
```

```
    ptr = next ;
  }
}

/* ======================================================
   routine: nfree_libc_float_list_list()
   ====================================================== */

void nfree_libc_float_list_list(struct libc_float_list_list **pptr)
{ struct libc_float_list_list *ptr= *pptr;

  struct libc_float_list_list *next;

  while (ptr!=NULL) {
    free_libc_float_list_rec(ptr->v_list);
    next = ptr->next ;
    free_mb_ptr((void *) ptr,1);
#if DMP_MEM_ANY
    dmp_libc_float_list_list_msize_dec(8);
    dmp_libc_mem_size_dec(8);
#endif
    ptr = next ;
  }
  *pptr = NULL;
}

/* ======================================================
   routine:  free_libc_define_rec()
   ====================================================== */

void  free_libc_define_rec(struct libc_define_rec *ptr)
{
  struct libc_define_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr->name);
    free_text_buffer(ptr->object);
    free_text_buffer(ptr->type);
    next = ptr->next ;
    free_mb_ptr((void *) ptr,3);
#if DMP_MEM_ANY
    dmp_libc_define_rec_msize_dec(16);
    dmp_libc_mem_size_dec(16);
#endif
    ptr = next ;
  }
}

/* ======================================================
   routine: nfree_libc_define_rec()
   ====================================================== */

void nfree_libc_define_rec(struct libc_define_rec **pptr)
{ struct libc_define_rec *ptr= *pptr;

  struct libc_define_rec *next;
```

```
  while (ptr!=NULL) {
    free_text_buffer(ptr->name);
    free_text_buffer(ptr->object);
    free_text_buffer(ptr->type);
    next = ptr->next ;
    free_mb_ptr((void *) ptr,3);
#if DMP_MEM_ANY
    dmp_libc_define_rec_msize_dec(16);
    dmp_libc_mem_size_dec(16);
#endif
    ptr = next ;
  }
  *pptr = NULL;
}

/* ============================================================
   routine:  free_libc_cell_area_rec()
   ============================================================ */

void  free_libc_cell_area_rec(struct libc_cell_area_rec *ptr)
{
  struct libc_cell_area_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr-> area_name);
    next = ptr->next ;
    free_mb_ptr((void *) ptr,2);
#if DMP_MEM_ANY
    dmp_libc_cell_area_rec_msize_dec(12);
    dmp_libc_mem_size_dec(12);
#endif
    ptr = next ;
  }
}

/* ============================================================
   routine: nfree_libc_cell_area_rec()
   ============================================================ */

void nfree_libc_cell_area_rec(struct libc_cell_area_rec **pptr)
{ struct libc_cell_area_rec *ptr= *pptr;

  struct libc_cell_area_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr->area_name);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,2);
#if DMP_MEM_ANY
    dmp_libc_cell_area_rec_msize_dec(12);
    dmp_libc_mem_size_dec(12);
#endif
    ptr = next ;
  }
  *pptr = NULL;
}
```

```
/* ========================================================
   routine:  free_libc_k_factor_rec()
   ======================================================== */

void  free_libc_k_factor_rec(struct libc_k_factor_rec *ptr)
{ int i0;
  struct libc_k_factor_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr->kf_name);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,29);
#if DMP_MEM_ANY
    dmp_libc_k_factor_rec_msize_dec(548);
    dmp_libc_mem_size_dec(548);
#endif
    ptr = next ;
  }
}

/* ========================================================
   routine: nfree_libc_k_factor_rec()
   ======================================================== */

void nfree_libc_k_factor_rec(struct libc_k_factor_rec **pptr)
{ struct libc_k_factor_rec *ptr= *pptr;
 int i0;
  struct libc_k_factor_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr-> kf_name);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,29);
#if DMP_MEM_ANY
    dmp_libc_k_factor_rec_msize_dec(548);
    dmp_libc_mem_size_dec(548);
#endif
    ptr = next ;
  }
  *pptr = NULL;
}

/* ========================================================
   routine:  free_libc_lib_rec()
   ======================================================== */

void  free_libc_lib_rec(struct libc_lib_rec *ptr)
{ int i0;
  if (ptr==NULL) return;
    free_text_buffer(ptr->lib_name);
    free_text_buffer(ptr->aux_no_pulldown_pin_property);
    free_text_buffer(ptr->bus_naming_style);
    free_text_buffer(ptr-> comment);
    free_any_unit_rec(ptr-> current_unit);
    free_text_buffer(ptr->date);
    free_any_unit_rec(ptr->leakage_power_unit);
    free_text_buffer(ptr->no_pulldown_pin_property);
```

```
    free_any_unit_rec(ptr->power_unit);
    free_text_buffer(ptr->preferred_output_pad_slew_rate_control);
    free_text_buffer(ptr->preferred_output_pad_voltage);
    free_text_buffer(ptr->preferred_input_pad_voltage);
    free_any_unit_rec(ptr->pulling_resistance_unit);
    free_text_buffer(ptr->revision);
    free_any_unit_rec(ptr-> time_unit);
    free_text_buffer(ptr->unconnected_pin_property);
    free_any_unit_rec(ptr->voltage_unit);
    free_libc_name_list_rec(ptr->default_connection_class);
    free_text_buffer(ptr->default_operating_conditions);
    free_text_buffer(ptr->default_wire_load);
    free_text_buffer(ptr-> default_wire_load_selection);
    free_libc_k_factor_rec(ptr->k_factor);
    free_libc_k_factor_rec(ptr->sc_k_factor);
    free_any_unit_rec(ptr->capacitive_load_unit);
    free_libc_define_rec(ptr->define);
    free_libc_cell_area_rec(ptr->define_cell_area);
    free_float_buffer(ptr->piece_define);
    free_libc_name_list_rec(ptr-> routing_layers);
    free_text_buffer(ptr-> technology);
    free_libc_lu_table_template_rec(ptr->lut_template);
    free_libc_lu_table_template_rec(ptr->plut_template);
    free_libc_operating_condition_rec(ptr->operating_cond);
    free_libc_power_supply_rec(ptr-> power_supply);
    free_libc_table_val_rec(ptr->rise_tr_table);
    free_libc_table_val_rec(ptr->fall_tr_table);
    free_libc_timing_range_rec(ptr->timing_range);
    free_libc_type_rec(ptr-> type);
    free_libc_wire_load_rec(ptr-> wire_load);
    free_libc_wire_load_selection_rec(ptr->wire_load_selection);
    free_libc_cell_rec(ptr-> cells);
    free_libc_define_value_rec(ptr->def_val);
    free_mb_ptr((void *) ptr,28);
#if DMP_MEM_ANY
    dmp_libc_lib_rec_msize_dec(392);
    dmp_libc_mem_size_dec(392);
#endif
}

/* ========================================================
   routine: nfree_libc_lib_rec()
   ======================================================== */

void nfree_libc_lib_rec(struct libc_lib_rec **pptr)
{ struct libc_lib_rec *ptr= *pptr;
 int i0;
  if (ptr==NULL) return;
    free_text_buffer(ptr->lib_name);
    free_text_buffer(ptr->aux_no_pulldown_pin_property);
    free_text_buffer(ptr-> bus_naming_style);
    free_text_buffer(ptr->comment);
    free_any_unit_rec(ptr->current_unit);
    free_text_buffer(ptr->date);
    free_any_unit_rec(ptr->leakage_power_unit);
    free_text_buffer(ptr-> no_pulldown_pin_property);
    free_any_unit_rec(ptr->power_unit);
```

```
    free_text_buffer(ptr->preferred_output_pad_slew_rate_control);
    free_text_buffer(ptr-> preferred_output_pad_voltage);
    free_text_buffer(ptr->preferred_input_pad_voltage);
    free_any_unit_rec(ptr->pulling_resistance_unit);
    free_text_buffer(ptr-> revision);
    free_any_unit_rec(ptr->time_unit);
    free_text_buffer(ptr->unconnected_pin_property);
    free_any_unit_rec(ptr->voltage_unit);
    free_libc_name_list_rec(ptr-> default_connection_class);
    free_text_buffer(ptr->default_operating_conditions);
    free_text_buffer(ptr->default_wire_load);
    free_text_buffer(ptr-> default_wire_load_selection);
    free_libc_k_factor_rec(ptr-> k_factor);
    free_libc_k_factor_rec(ptr->sc_k_factor);
    free_any_unit_rec(ptr->capacitive_load_unit);
    free_libc_define_rec(ptr-> define);
    free_libc_cell_area_rec(ptr->define_cell_area);
    free_float_buffer(ptr-> piece_define);
    free_libc_name_list_rec(ptr->routing_layers);
    free_text_buffer(ptr->technology);
    free_libc_lu_table_template_rec(ptr-> lut_template);
    free_libc_lu_table_template_rec(ptr->plut_template);
    free_libc_operating_condition_rec(ptr->operating_cond);
    free_libc_power_supply_rec(ptr->power_supply);
    free_libc_table_val_rec(ptr->rise_tr_table);
    free_libc_table_val_rec(ptr->fall_tr_table);
    free_libc_timing_range_rec(ptr-> timing_range);
    free_libc_type_rec(ptr->type);
    free_libc_wire_load_rec(ptr->wire_load);
    free_libc_wire_load_selection_rec(ptr->wire_load_selection);
    free_libc_cell_rec(ptr->cells);
    free_libc_define_value_rec(ptr->def_val);
    free_mb_ptr((void *) ptr,28);
#if DMP_MEM_ANY
    dmp_libc_lib_rec_msize_dec(392);
    dmp_libc_mem_size_dec(392);
#endif
  *pptr = NULL;
}

/* ========================================================
   routine:  free_libc_lu_table_template_rec()
   ======================================================== */

void  free_libc_lu_table_template_rec(struct libc_lu_table_template_rec *ptr)
{ int i0;
  struct libc_lu_table_template_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr-> tt_name);
    free_float_buffer(ptr->index_1);
    free_float_buffer(ptr-> index_2);
    free_float_buffer(ptr->index_3);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,7);
#if DMP_MEM_ANY
    dmp_libc_lu_table_template_rec_msize_dec(32);
```

```
    dmp_libc_mem_size_dec(32);
#endif
    ptr = next ;
  }
}

/* ======================================================
   routine: nfree_libc_lu_table_template_rec()
   ====================================================== */

void nfree_libc_lu_table_template_rec(struct libc_lu_table_template_rec
**pptr)
{ struct libc_lu_table_template_rec *ptr= *pptr;
 int i0;
  struct libc_lu_table_template_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr->tt_name);
    free_float_buffer(ptr->index_1);
    free_float_buffer(ptr-> index_2);
    free_float_buffer(ptr->index_3);
    next = ptr->next ;
    free_mb_ptr((void *) ptr,7);
#if DMP_MEM_ANY
    dmp_libc_lu_table_template_rec_msize_dec(32);
    dmp_libc_mem_size_dec(32);
#endif
    ptr = next ;
  }
  *pptr = NULL;
}

/* ======================================================
   routine:  free_libc_oc_power_rail_rec()
   ====================================================== */

void  free_libc_oc_power_rail_rec(struct libc_oc_power_rail_rec *ptr)
{
  struct libc_oc_power_rail_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr-> power_supply);
    next = ptr->next ;
    free_mb_ptr((void *) ptr,2);
#if DMP_MEM_ANY
    dmp_libc_oc_power_rail_rec_msize_dec(12);
    dmp_libc_mem_size_dec(12);
#endif
    ptr = next ;
  }
}

/* ======================================================
   routine: nfree_libc_oc_power_rail_rec()
   ====================================================== */

void nfree_libc_oc_power_rail_rec(struct libc_oc_power_rail_rec **pptr)
```

```
{ struct libc_oc_power_rail_rec *ptr= *pptr;

  struct libc_oc_power_rail_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr->power_supply);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,2);
#if DMP_MEM_ANY
    dmp_libc_oc_power_rail_rec_msize_dec(12);
    dmp_libc_mem_size_dec(12);
#endif
    ptr = next ;
  }
  *pptr = NULL;
}

/* ======================================================
   routine:  free_libc_operating_condition_rec()
   ====================================================== */

void  free_libc_operating_condition_rec(struct libc_operating_condition_rec
*ptr)
{
  struct libc_operating_condition_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr->oc_name);
    free_libc_oc_power_rail_rec(ptr->power_rail);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,6);
#if DMP_MEM_ANY
    dmp_libc_operating_condition_rec_msize_dec(28);
    dmp_libc_mem_size_dec(28);
#endif
    ptr = next ;
  }
}

/* ======================================================
   routine: nfree_libc_operating_condition_rec()
   ====================================================== */

void nfree_libc_operating_condition_rec(struct libc_operating_condition_rec
**pptr)
{ struct libc_operating_condition_rec *ptr= *pptr;

  struct libc_operating_condition_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr-> oc_name);
    free_libc_oc_power_rail_rec(ptr-> power_rail);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,6);
#if DMP_MEM_ANY
    dmp_libc_operating_condition_rec_msize_dec(28);
    dmp_libc_mem_size_dec(28);
```

```
#endif
    ptr = next ;
  }
  *pptr = NULL;
}

/* ======================================================
   routine:  free_libc_power_supply_rec()
   ====================================================== */

void  free_libc_power_supply_rec(struct libc_power_supply_rec *ptr)
{
  struct libc_power_supply_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr->ps_name);
    free_text_buffer(ptr->default_ps);
    free_libc_oc_power_rail_rec(ptr-> power_rail);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,3);
#if DMP_MEM_ANY
    dmp_libc_power_supply_rec_msize_dec(16);
    dmp_libc_mem_size_dec(16);
#endif
    ptr = next ;
  }
}

/* ======================================================
   routine: nfree_libc_power_supply_rec()
   ====================================================== */

void nfree_libc_power_supply_rec(struct libc_power_supply_rec **pptr)
{ struct libc_power_supply_rec *ptr= *pptr;

  struct libc_power_supply_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr->ps_name);
    free_text_buffer(ptr->default_ps);
    free_libc_oc_power_rail_rec(ptr->power_rail);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,3);
#if DMP_MEM_ANY
    dmp_libc_power_supply_rec_msize_dec(16);
    dmp_libc_mem_size_dec(16);
#endif
    ptr = next ;
  }
  *pptr = NULL;
}

/* ======================================================
   routine:  free_libc_timing_range_rec()
   ====================================================== */

void  free_libc_timing_range_rec(struct libc_timing_range_rec *ptr)
```

```
{
  struct libc_timing_range_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr->tr_name);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,3);
#if DMP_MEM_ANY
    dmp_libc_timing_range_rec_msize_dec(16);
    dmp_libc_mem_size_dec(16);
#endif
    ptr = next ;
  }
}

/* ======================================================
   routine: nfree_libc_timing_range_rec()
   ====================================================== */

void nfree_libc_timing_range_rec(struct libc_timing_range_rec **pptr)
{ struct libc_timing_range_rec *ptr= *pptr;

  struct libc_timing_range_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr->tr_name);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,3);
#if DMP_MEM_ANY
    dmp_libc_timing_range_rec_msize_dec(16);
    dmp_libc_mem_size_dec(16);
#endif
    ptr = next ;
  }
  *pptr = NULL;
}

/* ======================================================
   routine:  free_libc_type_rec()
   ====================================================== */

void  free_libc_type_rec(struct libc_type_rec *ptr)
{
  struct libc_type_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr-> type_name);
    next = ptr->next ;
    free_mb_ptr((void *) ptr,5);
#if DMP_MEM_ANY
    dmp_libc_type_rec_msize_dec(24);
    dmp_libc_mem_size_dec(24);
#endif
    ptr = next ;
  }
}
```

```
/* ======================================================
   routine: nfree_libc_type_rec()
   ====================================================== */

void nfree_libc_type_rec(struct libc_type_rec **pptr)
{ struct libc_type_rec *ptr= *pptr;

  struct libc_type_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr-> type_name);
    next = ptr->next ;
    free_mb_ptr((void *) ptr,5);
#if DMP_MEM_ANY
    dmp_libc_type_rec_msize_dec(24);
    dmp_libc_mem_size_dec(24);
#endif
    ptr = next ;
  }
  *pptr = NULL;
}

/* ======================================================
   routine:  free_libc_fanout_length_rec()
   ====================================================== */

void  free_libc_fanout_length_rec(struct libc_fanout_length_rec *ptr)
{
  struct libc_fanout_length_rec *next;

  while (ptr!=NULL) {
     next = ptr->next ;
    free_mb_ptr((void *) ptr,8);
#if DMP_MEM_ANY
    dmp_libc_fanout_length_rec_msize_dec(36);
    dmp_libc_mem_size_dec(36);
#endif
    ptr = next ;
  }
}

/* ======================================================
   routine: nfree_libc_fanout_length_rec()
   ====================================================== */

void nfree_libc_fanout_length_rec(struct libc_fanout_length_rec **pptr)
{ struct libc_fanout_length_rec *ptr= *pptr;

  struct libc_fanout_length_rec *next;

  while (ptr!=NULL) {
     next = ptr->next ;
    free_mb_ptr((void *) ptr,8);
#if DMP_MEM_ANY
    dmp_libc_fanout_length_rec_msize_dec(36);
    dmp_libc_mem_size_dec(36);
#endif
```

```
    ptr = next ;
  }
  *pptr = NULL;
}

/* ======================================================
   routine:  free_libc_wire_load_rec()
   ====================================================== */

void  free_libc_wire_load_rec(struct libc_wire_load_rec *ptr)
{
  struct libc_wire_load_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr->wl_name);
    free_libc_fanout_length_rec(ptr->fanout_length);
    next = ptr->next ;
    free_mb_ptr((void *) ptr,6);
#if DMP_MEM_ANY
    dmp_libc_wire_load_rec_msize_dec(28);
    dmp_libc_mem_size_dec(28);
#endif
    ptr = next ;
  }
}

/* ======================================================
   routine: nfree_libc_wire_load_rec()
   ====================================================== */

void nfree_libc_wire_load_rec(struct libc_wire_load_rec **pptr)
{ struct libc_wire_load_rec *ptr= *pptr;

  struct libc_wire_load_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr->wl_name);
    free_libc_fanout_length_rec(ptr->fanout_length);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,6);
#if DMP_MEM_ANY
    dmp_libc_wire_load_rec_msize_dec(28);
    dmp_libc_mem_size_dec(28);
#endif
    ptr = next ;
  }
  *pptr = NULL;
}

/* ======================================================
   routine:  free_libc_wire_load_from_area_rec()
   ====================================================== */

void  free_libc_wire_load_from_area_rec(struct libc_wire_load_from_area_rec
*ptr)
{
  struct libc_wire_load_from_area_rec *next;
```

```
  while (ptr!=NULL) {
     next = ptr->next ;
    free_mb_ptr((void *) ptr,3);
#if DMP_MEM_ANY
    dmp_libc_wire_load_from_area_rec_msize_dec(16);
    dmp_libc_mem_size_dec(16);
#endif
    ptr = next ;
  }
}

/* ======================================================
   routine: nfree_libc_wire_load_from_area_rec()
   ====================================================== */

void nfree_libc_wire_load_from_area_rec(struct libc_wire_load_from_area_rec
**pptr)
{ struct libc_wire_load_from_area_rec *ptr= *pptr;

  struct libc_wire_load_from_area_rec *next;

  while (ptr!=NULL) {
     next = ptr->next ;
    free_mb_ptr((void *) ptr,3);
#if DMP_MEM_ANY
    dmp_libc_wire_load_from_area_rec_msize_dec(16);
    dmp_libc_mem_size_dec(16);
#endif
    ptr = next ;
  }
  *pptr = NULL;
}

/* ======================================================
   routine:  free_libc_wire_load_selection_rec()
   ====================================================== */

void  free_libc_wire_load_selection_rec(struct libc_wire_load_selection_rec
*ptr)
{
  struct libc_wire_load_selection_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr->wls_name);
    free_libc_wire_load_from_area_rec(ptr->area_table);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,2);
#if DMP_MEM_ANY
    dmp_libc_wire_load_selection_rec_msize_dec(12);
    dmp_libc_mem_size_dec(12);
#endif
    ptr = next ;
  }
}

/* ======================================================
```

```
   routine: nfree_libc_wire_load_selection_rec()
   ========================================================== */

void nfree_libc_wire_load_selection_rec(struct libc_wire_load_selection_rec
**pptr)
{ struct libc_wire_load_selection_rec *ptr= *pptr;

  struct libc_wire_load_selection_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr->wls_name);
    free_libc_wire_load_from_area_rec(ptr->area_table);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,2);
#if DMP_MEM_ANY
    dmp_libc_wire_load_selection_rec_msize_dec(12);
    dmp_libc_mem_size_dec(12);
#endif
    ptr = next ;
  }
  *pptr = NULL;
}

/* ==========================================================
   routine:  free_libc_cell_rec()
   ========================================================== */

void  free_libc_cell_rec(struct libc_cell_rec *ptr)
{
  struct libc_cell_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr->cell_name);
    free_text_buffer(ptr->cell_footprint);
    free_libc_bool_opr_rec(ptr->contention_condition);
    free_text_buffer(ptr->geometry_print);
    free_text_buffer(ptr->scan_group);
    free_text_buffer(ptr->single_bit_degenerate);
    free_text_buffer(ptr->vhdl_name);
    free_libc_name_list_list(ptr->pin_equal);
    free_libc_name_list_list(ptr->pin_opposite);
    free_libc_pin_rec(ptr->pins);
    free_libc_internal_power(ptr->internal_power_c);
    free_libc_leakage_power(ptr->leakage_power);
    free_libc_ff_latch_rec(ptr->ff_latch);
    free_libc_memory_rec(ptr-> memory);
    free_libc_routing_track_rec(ptr->routing_track);
    free_libc_define_value_rec(ptr-> def_val);
    next = ptr->next ;
    free_mb_ptr((void *) ptr,23);
#if DMP_MEM_ANY
    dmp_libc_cell_rec_msize_dec(132);
    dmp_libc_mem_size_dec(132);
#endif
    ptr = next ;
  }
}
```

```
/* ======================================================
   routine: nfree_libc_cell_rec()
   ====================================================== */

void nfree_libc_cell_rec(struct libc_cell_rec **pptr)
{ struct libc_cell_rec *ptr= *pptr;

  struct libc_cell_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr->cell_name);
    free_text_buffer(ptr->cell_footprint);
    free_libc_bool_opr_rec(ptr->contention_condition);
    free_text_buffer(ptr->geometry_print);
    free_text_buffer(ptr->scan_group);
    free_text_buffer(ptr->single_bit_degenerate);
    free_text_buffer(ptr-> vhdl_name);
    free_libc_name_list_list(ptr->pin_equal);
    free_libc_name_list_list(ptr->pin_opposite);
    free_libc_pin_rec(ptr-> pins);
    free_libc_internal_power(ptr-> internal_power_c);
    free_libc_leakage_power(ptr->leakage_power);
    free_libc_ff_latch_rec(ptr->ff_latch);
    free_libc_memory_rec(ptr->memory);
    free_libc_routing_track_rec(ptr->routing_track);
    free_libc_define_value_rec(ptr->def_val);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,23);
#if DMP_MEM_ANY
    dmp_libc_cell_rec_msize_dec(132);
    dmp_libc_mem_size_dec(132);
#endif
    ptr = next ;
  }
  *pptr = NULL;
}

/* ======================================================
   routine:  free_libc_memory_write_rec()
   ====================================================== */

void  free_libc_memory_write_rec(struct libc_memory_write_rec *ptr)
{
  if (ptr==NULL) return;
    free_text_buffer(ptr-> address);
    free_text_buffer(ptr->clock_on);
    free_text_buffer(ptr-> enable);
    free_mb_ptr((void *) ptr,2);
#if DMP_MEM_ANY
    dmp_libc_memory_write_rec_msize_dec(12);
    dmp_libc_mem_size_dec(12);
#endif
}

/* ======================================================
   routine: nfree_libc_memory_write_rec()
```

```
   ========================================================== */

void nfree_libc_memory_write_rec(struct libc_memory_write_rec **pptr)
{ struct libc_memory_write_rec *ptr= *pptr;

  if (ptr==NULL) return;
    free_text_buffer(ptr->address);
    free_text_buffer(ptr-> clock_on);
    free_text_buffer(ptr->enable);
    free_mb_ptr((void *) ptr,2);
#if DMP_MEM_ANY
    dmp_libc_memory_write_rec_msize_dec(12);
    dmp_libc_mem_size_dec(12);
#endif
  *pptr = NULL;
}

struct libc_memory_write_rec * copy_libc_memory_write_rec(struct
libc_memory_write_rec *sou)
{ struct libc_memory_write_rec *tar,*head,*prev;

  if (sou==NULL) return(NULL);
  prev = NULL;
  head = tar = new_libc_memory_write_rec();
    tar->address    = (char *) copy_string(sou->address);
    tar->clock_on   = (char *) copy_string(sou->clock_on);
    tar->enable     = (char *) copy_string(sou->enable);

    return(head);
}

/* ==========================================================
   routine:  free_libc_bool_opr_rec()
   ========================================================== */

void  free_libc_bool_opr_rec(struct libc_bool_opr_rec *ptr)
{
  struct libc_bool_opr_rec *next;

  while (ptr!=NULL) {
    switch (ptr-> type) {
      case ID_B :
           free_text_buffer(ptr-> u.id_name);
           break;

      default :
           break;

    }
    free_libc_bool_opr_rec(ptr->L);
     next = ptr->R ;
    free_mb_ptr((void *) ptr,3);
#if DMP_MEM_ANY
    dmp_libc_bool_opr_rec_msize_dec(16);
    dmp_libc_mem_size_dec(16);
#endif
    ptr = next ;
```

```
  }
}

/* ========================================================
   routine: nfree_libc_bool_opr_rec()
   ======================================================== */

void nfree_libc_bool_opr_rec(struct libc_bool_opr_rec **pptr)
{ struct libc_bool_opr_rec *ptr= *pptr;

  struct libc_bool_opr_rec *next;

  while (ptr!=NULL) {
    switch (ptr->type) {
      case ID_B :
           free_text_buffer(ptr->u.id_name);
           break;

      default :
           break;

    }
    free_libc_bool_opr_rec(ptr->L);
    next = ptr->R ;
    free_mb_ptr((void *) ptr,3);
#if DMP_MEM_ANY
    dmp_libc_bool_opr_rec_msize_dec(16);
    dmp_libc_mem_size_dec(16);
#endif
    ptr = next ;
  }
  *pptr = NULL;
}

struct libc_bool_opr_rec * copy_libc_bool_opr_rec(struct libc_bool_opr_rec
*sou)
{ struct libc_bool_opr_rec *tar,*head,*prev;

  if (sou==NULL) return(NULL);
  prev = NULL;
  head = tar = new_libc_bool_opr_rec();
  while (sou!=NULL) {
    memcpy((char *)tar,(char *)sou,sizeof(struct libc_bool_opr_rec));
    tar->type       = sou->type;
    switch (sou->type) {
      case ID_B :
           tar->u.id_name    = (char *) copy_string(sou->u.id_name);
           break;

      default :
           tar->u.value      = sou->u.value;
           break;

    }
    tar->L          = (struct libc_bool_opr_rec *)
copy_libc_bool_opr_rec(sou->L);
    sou  = sou->R ;
```

```
    if (sou!=NULL)
    { prev = tar;
      tar->R = new_libc_bool_opr_rec();
      tar = tar->R;
    }
  }
    return(head);
}


/* ======================================================
   routine:  free_libc_pin_rec()
   ====================================================== */

void  free_libc_pin_rec(struct libc_pin_rec *ptr)
{
  struct libc_pin_rec *next;

  while (ptr!=NULL) {
    free_libc_name_list_rec(ptr->pin_name);
    free_libc_name_list_rec(ptr->members);
    free_libc_name_list_rec(ptr->connection_class);
    free_libc_bool_opr_rec(ptr-> function);
    free_libc_name_list_rec(ptr->input_map);
    free_text_buffer(ptr->input_signal_level);
    free_text_buffer(ptr->input_voltage);
    free_text_buffer(ptr->internal_node);
    free_text_buffer(ptr->output_signal_level);
    free_text_buffer(ptr-> output_voltage);
    free_libc_bool_opr_rec(ptr-> state_function);
    free_libc_bool_opr_rec(ptr->three_state);
    free_text_buffer(ptr->vhdl_name);
    free_text_buffer(ptr->wired_connection_class);
    free_libc_bool_opr_rec(ptr->x_function);
    free_text_buffer(ptr->address_of_memory_read);
    free_libc_memory_write_rec(ptr->memory_write);
    free_libc_timing_rec(ptr->timing);
    free_libc_min_pulse_width_rec(ptr->min_pulse_width);
    free_libc_minimum_period_rec(ptr->minimum_period);
    free_libc_internal_power(ptr->internal_power);
    free_libc_define_value_rec(ptr->def_val);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,27);
#if DMP_MEM_ANY
    dmp_libc_pin_rec_msize_dec(276);
    dmp_libc_mem_size_dec(276);
#endif
    ptr = next ;
  }
}

/* ======================================================
   routine: nfree_libc_pin_rec()
   ====================================================== */

void nfree_libc_pin_rec(struct libc_pin_rec **pptr)
{ struct libc_pin_rec *ptr= *pptr;
```

```
  struct libc_pin_rec *next;

  while (ptr!=NULL) {
    free_libc_name_list_rec(ptr->pin_name);
    free_libc_name_list_rec(ptr->members);
    free_libc_name_list_rec(ptr-> connection_class);
    free_libc_bool_opr_rec(ptr->function);
    free_libc_name_list_rec(ptr->input_map);
    free_text_buffer(ptr->input_signal_level);
    free_text_buffer(ptr->input_voltage);
    free_text_buffer(ptr-> internal_node);
    free_text_buffer(ptr->output_signal_level);
    free_text_buffer(ptr->output_voltage);
    free_libc_bool_opr_rec(ptr->state_function);
    free_libc_bool_opr_rec(ptr->three_state);
    free_text_buffer(ptr->vhdl_name);
    free_text_buffer(ptr->wired_connection_class);
    free_libc_bool_opr_rec(ptr->x_function);
    free_text_buffer(ptr-> address_of_memory_read);
    free_libc_memory_write_rec(ptr->memory_write);
    free_libc_timing_rec(ptr->timing);
    free_libc_min_pulse_width_rec(ptr-> min_pulse_width);
    free_libc_minimum_period_rec(ptr->minimum_period);
    free_libc_internal_power(ptr->internal_power);
    free_libc_define_value_rec(ptr->def_val);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,27);
#if DMP_MEM_ANY
    dmp_libc_pin_rec_msize_dec(276);
    dmp_libc_mem_size_dec(276);
#endif
    ptr = next ;
  }
  *pptr = NULL;
}

struct libc_pin_rec * copy_libc_pin_rec(struct libc_pin_rec *sou)
{ struct libc_pin_rec *tar,*head,*prev;

  if (sou==NULL) return(NULL);
  prev = NULL;
  head = tar = new_libc_pin_rec();
  while (sou!=NULL) {
    tar->pin_name   = (struct libc_name_list_rec *)
copy_libc_name_list_rec(sou->pin_name);
    tar->_current_cell = sou->_current_cell;
    tar->members    = (struct libc_name_list_rec *)
copy_libc_name_list_rec(sou->members);
    tar->is_bus     = sou->is_bus;
    tar->_bus_type  = sou->_bus_type;
    tar->capacitance = sou->capacitance;
    tar->clock      = sou->clock;
    tar->clock_gate_enable_pin = sou->clock_gate_enable_pin;
    tar->connection_class = (struct libc_name_list_rec *)
copy_libc_name_list_rec(sou->connection_class);
    tar->direction  = sou->direction;
```

```
    tar->dont_false = sou->dont_false;
    tar->drive_current = sou->drive_current;
    tar->drive_type = sou->drive_type;
    tar->emitter_count = sou->emitter_count;
    tar->fall_current_slop_after_threshold = sou-
>fall_current_slop_after_threshold;
    tar->fall_current_slop_before_threshold = sou-
>fall_current_slop_before_threshold;
    tar->fall_time_after_threshold = sou->fall_time_after_threshold;
    tar->fall_time_before_threshold = sou->fall_time_before_threshold;
    tar->fall_wor_emitter = sou->fall_wor_emitter;
    tar->fall_wor_intercept = sou->fall_wor_intercept;
    tar->fanout_load = sou->fanout_load;
    tar->function   = (struct libc_bool_opr_rec *)
copy_libc_bool_opr_rec(sou->function);
    tar->hysteresis = sou->hysteresis;
    tar->input_map  = (struct libc_name_list_rec *)
copy_libc_name_list_rec(sou->input_map);
    tar->input_signal_level = (char *) copy_string(sou->input_signal_level);
    tar->input_voltage = (char *) copy_string(sou->input_voltage);
    tar->internal_node = (char *) copy_string(sou->internal_node);
    tar->inverted_output = sou->inverted_output;
    tar->is_pad      = sou->is_pad;
    tar->max_fanout = sou->max_fanout;
    tar->max_transition = sou->max_transition;
    tar->max_capacitance = sou->max_capacitance;
    tar->min_fanout = sou->min_fanout;
    tar->min_transition = sou->min_transition;
    tar->min_capacitance = sou->min_capacitance;
    tar->multicell_pad_pin = sou->multicell_pad_pin;
    tar->multiple_drivers_legal = sou->multiple_drivers_legal;
    tar->nextstate_type = sou->nextstate_type;
    tar->output_signal_level = (char *) copy_string(sou-
>output_signal_level);
    tar->output_voltage = (char *) copy_string(sou->output_voltage);
    tar->pin_func_type = sou->pin_func_type;
    tar->pin_power  = sou->pin_power;
    tar->prefer_tied = sou->prefer_tied;
    tar->primary_output = sou->primary_output;
    tar->pulling_current = sou->pulling_current;
    tar->pulling_resistance = sou->pulling_resistance;
    tar->reference_capacitance = sou->reference_capacitance;
    tar->rise_current_slop_after_threshold = sou-
>rise_current_slop_after_threshold;
    tar->rise_current_slop_before_threshold = sou-
>rise_current_slop_before_threshold;
    tar->rise_time_after_threshold = sou->rise_time_after_threshold;
    tar->rise_time_before_threshold = sou->rise_time_before_threshold;
    tar->rise_wor_emitter = sou->rise_wor_emitter;
    tar->rise_wor_intercept = sou->rise_wor_intercept;
    tar->slew_control = sou->slew_control;
    tar->state_function = (struct libc_bool_opr_rec *)
copy_libc_bool_opr_rec(sou->state_function);
    tar->three_state = (struct libc_bool_opr_rec *)
copy_libc_bool_opr_rec(sou->three_state);
    tar->vhdl_name  = (char *) copy_string(sou->vhdl_name);
    tar->wire_capacitance = sou->wire_capacitance;
```

```
    tar->wired_connection_class = (char *) copy_string(sou-
>wired_connection_class);
    tar->x_function = (struct libc_bool_opr_rec *)
copy_libc_bool_opr_rec(sou->x_function);
    tar->address_of_memory_read = (char *) copy_string(sou-
>address_of_memory_read);
    tar->memory_write = (struct libc_memory_write_rec *)
copy_libc_memory_write_rec(sou->memory_write);
    tar->timing     = (struct libc_timing_rec *) copy_libc_timing_rec(sou-
>timing);
    tar->min_pulse_width = (struct libc_min_pulse_width_rec *)
copy_libc_min_pulse_width_rec(sou->min_pulse_width);
    tar->minimum_period = (struct libc_minimum_period_rec *)
copy_libc_minimum_period_rec(sou->minimum_period);
    tar->internal_power = (struct libc_internal_power *)
copy_libc_internal_power(sou->internal_power);
    tar->def_val    = (struct libc_define_value_rec *)
copy_libc_define_value_rec(sou->def_val);
    tar->pin_type   = sou->pin_type;
    sou  = sou->next ;
    if (sou!=NULL)
    { prev = tar;
      tar->next = new_libc_pin_rec();
      tar = tar->next;
    }
  }
    return(head);
}


/* ============================================================
   routine:  free_libc_ff_latch_rec()
   ============================================================ */

void  free_libc_ff_latch_rec(struct libc_ff_latch_rec *ptr)
{
  struct libc_ff_latch_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr->Q_name);
    free_text_buffer(ptr->QN_name);
    free_libc_bool_opr_rec(ptr->clear);
    free_libc_bool_opr_rec(ptr->preset);
    free_libc_bool_opr_rec(ptr-> clock_on);
    free_libc_bool_opr_rec(ptr-> next_state);
    free_libc_bool_opr_rec(ptr->on_also);
    free_libc_bool_opr_rec(ptr->enable);
    free_libc_bool_opr_rec(ptr->data_in);
    free_libc_bool_opr_rec(ptr->force_00);
    free_libc_bool_opr_rec(ptr->force_01);
    free_libc_bool_opr_rec(ptr->force_10);
    free_libc_bool_opr_rec(ptr->force_11);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,15);
#if DMP_MEM_ANY
    dmp_libc_ff_latch_rec_msize_dec(64);
    dmp_libc_mem_size_dec(64);
```

```
#endif
    ptr = next ;
  }
}

/* ========================================================
   routine: nfree_libc_ff_latch_rec()
   ======================================================== */

void nfree_libc_ff_latch_rec(struct libc_ff_latch_rec **pptr)
{ struct libc_ff_latch_rec *ptr= *pptr;

  struct libc_ff_latch_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr->Q_name);
    free_text_buffer(ptr-> QN_name);
    free_libc_bool_opr_rec(ptr->clear);
    free_libc_bool_opr_rec(ptr->preset);
    free_libc_bool_opr_rec(ptr->clock_on);
    free_libc_bool_opr_rec(ptr-> next_state);
    free_libc_bool_opr_rec(ptr->on_also);
    free_libc_bool_opr_rec(ptr-> enable);
    free_libc_bool_opr_rec(ptr->data_in);
    free_libc_bool_opr_rec(ptr->force_00);
    free_libc_bool_opr_rec(ptr->force_01);
    free_libc_bool_opr_rec(ptr->force_10);
    free_libc_bool_opr_rec(ptr->force_11);
    next = ptr->next ;
    free_mb_ptr((void *) ptr,15);
#if DMP_MEM_ANY
    dmp_libc_ff_latch_rec_msize_dec(64);
    dmp_libc_mem_size_dec(64);
#endif
    ptr = next ;
  }
  *pptr = NULL;
}

/* ========================================================
   routine:  free_libc_internal_power()
   ======================================================== */

void  free_libc_internal_power(struct libc_internal_power *ptr)
{
  struct libc_internal_power *next;

  while (ptr!=NULL) {
    free_libc_name_list_rec(ptr->inputs);
    free_libc_name_list_rec(ptr->outputs);
    free_libc_name_list_rec(ptr->equal_or_opposite_output);
    free_text_buffer(ptr->power_level);
    free_libc_name_list_rec(ptr->related_pin);
    free_libc_bool_opr_rec(ptr-> when);
    free_libc_table_val_rec(ptr-> rise_power);
    free_libc_table_val_rec(ptr->fall_power);
    free_libc_table_val_rec(ptr->power);
```

```
     next = ptr->next ;
    free_mb_ptr((void *) ptr,9);
#if DMP_MEM_ANY
    dmp_libc_internal_power_msize_dec(40);
    dmp_libc_mem_size_dec(40);
#endif
    ptr = next ;
  }
}

/* ========================================================
   routine: nfree_libc_internal_power()
   ======================================================== */

void nfree_libc_internal_power(struct libc_internal_power **pptr)
{ struct libc_internal_power *ptr= *pptr;

  struct libc_internal_power *next;

  while (ptr!=NULL) {
    free_libc_name_list_rec(ptr-> inputs);
    free_libc_name_list_rec(ptr->outputs);
    free_libc_name_list_rec(ptr->equal_or_opposite_output);
    free_text_buffer(ptr->power_level);
    free_libc_name_list_rec(ptr->related_pin);
    free_libc_bool_opr_rec(ptr-> when);
    free_libc_table_val_rec(ptr->rise_power);
    free_libc_table_val_rec(ptr->fall_power);
    free_libc_table_val_rec(ptr->power);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,9);
#if DMP_MEM_ANY
    dmp_libc_internal_power_msize_dec(40);
    dmp_libc_mem_size_dec(40);
#endif
    ptr = next ;
  }
  *pptr = NULL;
}

struct libc_internal_power * copy_libc_internal_power(struct
libc_internal_power *sou)
{ struct libc_internal_power *tar,*head,*prev;

  if (sou==NULL) return(NULL);
  prev = NULL;
  head = tar = new_libc_internal_power();
  while (sou!=NULL) {
    tar->inputs     = (struct libc_name_list_rec *)
copy_libc_name_list_rec(sou->inputs);
    tar->outputs    = (struct libc_name_list_rec *)
copy_libc_name_list_rec(sou->outputs);
    tar->equal_or_opposite_output = (struct libc_name_list_rec *)
copy_libc_name_list_rec(sou->equal_or_opposite_output);
    tar->power_level = (char *) copy_string(sou->power_level);
    tar->related_pin = (struct libc_name_list_rec *)
copy_libc_name_list_rec(sou->related_pin);
```

```
    tar->when       = (struct libc_bool_opr_rec *)
copy_libc_bool_opr_rec(sou->when);
    tar->rise_power = (struct libc_table_val_rec *)
copy_libc_table_val_rec(sou->rise_power);
    tar->fall_power = (struct libc_table_val_rec *)
copy_libc_table_val_rec(sou->fall_power);
    tar->power      = (struct libc_table_val_rec *)
copy_libc_table_val_rec(sou->power);
    sou  = sou->next ;
    if (sou!=NULL)
    { prev = tar;
      tar->next = new_libc_internal_power();
      tar = tar->next;
    }
  }
    return(head);
}


/* ========================================================
   routine:  free_libc_leakage_power()
   ======================================================== */

void  free_libc_leakage_power(struct libc_leakage_power *ptr)
{
  struct libc_leakage_power *next;

  while (ptr!=NULL) {
    free_libc_bool_opr_rec(ptr-> when);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,2);
#if DMP_MEM_ANY
    dmp_libc_leakage_power_msize_dec(12);
    dmp_libc_mem_size_dec(12);
#endif
    ptr = next ;
  }
}

/* ========================================================
   routine: nfree_libc_leakage_power()
   ======================================================== */

void nfree_libc_leakage_power(struct libc_leakage_power **pptr)
{ struct libc_leakage_power *ptr= *pptr;

  struct libc_leakage_power *next;

  while (ptr!=NULL) {
    free_libc_bool_opr_rec(ptr->when);
    next = ptr->next ;
    free_mb_ptr((void *) ptr,2);
#if DMP_MEM_ANY
    dmp_libc_leakage_power_msize_dec(12);
    dmp_libc_mem_size_dec(12);
#endif
    ptr = next ;
```

```
  }
  *pptr = NULL;
}

struct libc_leakage_power * copy_libc_leakage_power(struct libc_leakage_power
*sou)
{ struct libc_leakage_power *tar,*head,*prev;

  if (sou==NULL) return(NULL);
  prev = NULL;
  head = tar = new_libc_leakage_power();
  while (sou!=NULL) {
    tar->when       = (struct libc_bool_opr_rec *)
copy_libc_bool_opr_rec(sou->when);
    tar->value      = sou->value;
    sou  = sou->next ;
    if (sou!=NULL)
    { prev = tar;
      tar->next = new_libc_leakage_power();
      tar = tar->next;
    }
  }
    return(head);
}


/* ========================================================
   routine:  free_libc_memory_rec()
   ======================================================== */

void  free_libc_memory_rec(struct libc_memory_rec *ptr)
{
  if (ptr==NULL) return;
    free_mb_ptr((void *) ptr,2);
#if DMP_MEM_ANY
    dmp_libc_memory_rec_msize_dec(12);
    dmp_libc_mem_size_dec(12);
#endif
}

/* ========================================================
   routine: nfree_libc_memory_rec()
   ======================================================== */

void nfree_libc_memory_rec(struct libc_memory_rec **pptr)
{ struct libc_memory_rec *ptr= *pptr;

  if (ptr==NULL) return;
    free_mb_ptr((void *) ptr,2);
#if DMP_MEM_ANY
    dmp_libc_memory_rec_msize_dec(12);
    dmp_libc_mem_size_dec(12);
#endif
  *pptr = NULL;
}

/* ========================================================
```

```
   routine:  free_libc_piece_value_rec()
   ========================================================== */

void  free_libc_piece_value_rec(struct libc_piece_value_rec *ptr)
{
  struct libc_piece_value_rec *next;

  while (ptr!=NULL) {
    next = ptr->next ;
    free_mb_ptr((void *) ptr,9);
#if DMP_MEM_ANY
    dmp_libc_piece_value_rec_msize_dec(40);
    dmp_libc_mem_size_dec(40);
#endif
    ptr = next ;
  }
}

/* ========================================================
   routine: nfree_libc_piece_value_rec()
   ========================================================= */

void nfree_libc_piece_value_rec(struct libc_piece_value_rec **pptr)
{ struct libc_piece_value_rec *ptr= *pptr;

  struct libc_piece_value_rec *next;

  while (ptr!=NULL) {
     next = ptr->next ;
    free_mb_ptr((void *) ptr,9);
#if DMP_MEM_ANY
    dmp_libc_piece_value_rec_msize_dec(40);
    dmp_libc_mem_size_dec(40);
#endif
    ptr = next ;
  }
  *pptr = NULL;
}

struct libc_piece_value_rec * copy_libc_piece_value_rec(struct
libc_piece_value_rec *sou)
{ struct libc_piece_value_rec *tar,*head,*prev;

  if (sou==NULL) return(NULL);
  prev = NULL;
  head = tar = new_libc_piece_value_rec();
  while (sou!=NULL) {
    tar->piece      = sou->piece;
    tar->fall_delay_intercept = sou->fall_delay_intercept;
    tar->fall_nonpaired_twin = sou->fall_nonpaired_twin;
    tar->fall_pin_resistance = sou->fall_pin_resistance;
    tar->fall_wire_resistance = sou->fall_wire_resistance;
    tar->rise_delay_intercept = sou->rise_delay_intercept;
    tar->rise_nonpaired_twin = sou->rise_nonpaired_twin;
    tar->rise_pin_resistance = sou->rise_pin_resistance;
    tar->rise_wire_resistance = sou->rise_wire_resistance;
    sou  = sou->next ;
```

```
    if (sou!=NULL)
    { prev = tar;
      tar->next = new_libc_piece_value_rec();
      tar = tar->next;
    }
  }
    return(head);
}


/* ========================================================
   routine:  free_libc_float_rec()
   ======================================================== */

void  free_libc_float_rec(struct libc_float_rec *ptr)
{
  if (ptr==NULL) return;
    free_mb_ptr((void *) ptr,0);
#if DMP_MEM_ANY
    dmp_libc_float_rec_msize_dec(4);
    dmp_libc_mem_size_dec(4);
#endif
}

/* ========================================================
   routine: nfree_libc_float_rec()
   ======================================================== */

void nfree_libc_float_rec(struct libc_float_rec **pptr)
{ struct libc_float_rec *ptr= *pptr;

  if (ptr==NULL) return;
    free_mb_ptr((void *) ptr,0);
#if DMP_MEM_ANY
    dmp_libc_float_rec_msize_dec(4);
    dmp_libc_mem_size_dec(4);
#endif
  *pptr = NULL;
}

struct libc_float_rec * copy_libc_float_rec(struct libc_float_rec *sou)
{ struct libc_float_rec *tar,*head,*prev;

  if (sou==NULL) return(NULL);
  prev = NULL;
  head = tar = new_libc_float_rec();
  memcpy((char *)tar,(char *)sou,sizeof(struct libc_float_rec));

  return(head);
}

/* ========================================================
   routine:  free_libc_table_val_rec()
   ======================================================== */

void  free_libc_table_val_rec(struct libc_table_val_rec *ptr)
{ int i0;
```

```
  if (ptr==NULL) return;
    free_float_buffer(ptr->index1);
    free_float_buffer(ptr->index2);
    free_float_buffer(ptr->index3);
    free_float_buffer(ptr-> values);
    free_mb_ptr((void *) ptr,5);
#if DMP_MEM_ANY
    dmp_libc_table_val_rec_msize_dec(24);
    dmp_libc_mem_size_dec(24);
#endif
}

/* ========================================================
   routine: nfree_libc_table_val_rec()
   ======================================================== */

void nfree_libc_table_val_rec(struct libc_table_val_rec **pptr)
{ struct libc_table_val_rec *ptr= *pptr;
 int i0;
  if (ptr==NULL) return;
    free_float_buffer(ptr-> index1);
    free_float_buffer(ptr->index2);
    free_float_buffer(ptr->index3);
    free_float_buffer(ptr-> values);
    free_mb_ptr((void *) ptr,5);
#if DMP_MEM_ANY
    dmp_libc_table_val_rec_msize_dec(24);
    dmp_libc_mem_size_dec(24);
#endif
  *pptr = NULL;
}

struct libc_table_val_rec * copy_libc_table_val_rec(struct libc_table_val_rec
*sou)
{ struct libc_table_val_rec *tar,*head,*prev;
  int i0;
  if (sou==NULL) return(NULL);
  prev = NULL;
  head = tar = new_libc_table_val_rec();
    tar->_tbl       = sou->_tbl;
    tar->index1     = get_float_buffer(sizeof_float_buffer(sou->index1));
    for (i0=0;i0<sizeof_float_buffer(sou->index1);i0++)
     tar->index1 [i0] = sou->index1[i0];
    tar->index2     = get_float_buffer(sizeof_float_buffer(sou->index2));
    for (i0=0;i0<sizeof_float_buffer(sou->index2);i0++)
     tar->index2 [i0] = sou->index2[i0];
    tar->index3     = get_float_buffer(sizeof_float_buffer(sou->index3));
    for (i0=0;i0<sizeof_float_buffer(sou->index3);i0++)
     tar->index3 [i0] = sou->index3[i0];
    tar->scalar_val = sou->scalar_val;
    tar->values     = get_float_buffer(sizeof_float_buffer(sou->values));
    for (i0=0;i0<sizeof_float_buffer(sou->values);i0++)
     tar->values [i0] = sou->values[i0];

    return(head);
}
```

```
/* ========================================================
   routine:  free_libc_timing_rec()
   ======================================================== */

void  free_libc_timing_rec(struct libc_timing_rec *ptr)
{
  struct libc_timing_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr-> t_name);
    free_libc_bool_opr_rec(ptr->related_bus_pins);
    free_libc_name_list_rec(ptr->related_pin);
    free_libc_bool_opr_rec(ptr->when);
    free_libc_bool_opr_rec(ptr->when_start);
    free_libc_bool_opr_rec(ptr-> when_end);
    free_libc_piece_value_rec(ptr->piecewise);
    free_libc_table_val_rec(ptr->cell_rise);
    free_libc_table_val_rec(ptr-> cell_fall);
    free_libc_table_val_rec(ptr->rise_propagation);
    free_libc_table_val_rec(ptr-> fall_propagation);
    free_libc_table_val_rec(ptr->rise_transition);
    free_libc_table_val_rec(ptr->fall_transition);
    free_libc_table_val_rec(ptr->rise_constraint);
    free_libc_table_val_rec(ptr->fall_constraint);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,22);
#if DMP_MEM_ANY
    dmp_libc_timing_rec_msize_dec(120);
    dmp_libc_mem_size_dec(120);
#endif
    ptr = next ;
  }
}

/* ========================================================
   routine: nfree_libc_timing_rec()
   ======================================================== */

void nfree_libc_timing_rec(struct libc_timing_rec **pptr)
{ struct libc_timing_rec *ptr= *pptr;

  struct libc_timing_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr-> t_name);
    free_libc_bool_opr_rec(ptr-> related_bus_pins);
    free_libc_name_list_rec(ptr->related_pin);
    free_libc_bool_opr_rec(ptr-> when);
    free_libc_bool_opr_rec(ptr->when_start);
    free_libc_bool_opr_rec(ptr->when_end);
    free_libc_piece_value_rec(ptr->piecewise);
    free_libc_table_val_rec(ptr-> cell_rise);
    free_libc_table_val_rec(ptr->cell_fall);
    free_libc_table_val_rec(ptr->rise_propagation);
    free_libc_table_val_rec(ptr-> fall_propagation);
    free_libc_table_val_rec(ptr-> rise_transition);
    free_libc_table_val_rec(ptr->fall_transition);
```

```
    free_libc_table_val_rec(ptr->rise_constraint);
    free_libc_table_val_rec(ptr-> fall_constraint);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,22);
#if DMP_MEM_ANY
    dmp_libc_timing_rec_msize_dec(120);
    dmp_libc_mem_size_dec(120);
#endif
    ptr = next ;
  }
  *pptr = NULL;
}

struct libc_timing_rec * copy_libc_timing_rec(struct libc_timing_rec *sou)
{ struct libc_timing_rec *tar,*head,*prev;

  if (sou==NULL) return(NULL);
  prev = NULL;
  head = tar = new_libc_timing_rec();
  while (sou!=NULL) {
    tar->t_name     = (char *) copy_string(sou->t_name);
    tar->_current_pin = sou->_current_pin;
    tar->edge_rate_sensitivity_f0 = sou->edge_rate_sensitivity_f0;
    tar->edge_rate_sensitivity_f1 = sou->edge_rate_sensitivity_f1;
    tar->edge_rate_sensitivity_r0 = sou->edge_rate_sensitivity_r0;
    tar->edge_rate_sensitivity_r1 = sou->edge_rate_sensitivity_r1;
    tar->fall_resistance = sou->fall_resistance;
    tar->rise_resistance = sou->rise_resistance;
    tar->intrinsic_fall = sou->intrinsic_fall;
    tar->intrinsic_rise = sou->intrinsic_rise;
    tar->related_bus_pins = (struct libc_bool_opr_rec *)
copy_libc_bool_opr_rec(sou->related_bus_pins);
    tar->related_output_pin = sou->related_output_pin;
    tar->related_pin = (struct libc_name_list_rec *)
copy_libc_name_list_rec(sou->related_pin);
    tar->slope_fall = sou->slope_fall;
    tar->slope_rise = sou->slope_rise;
    tar->timing_type = sou->timing_type;
    tar->timing_sense = sou->timing_sense;
    tar->when       = (struct libc_bool_opr_rec *)
copy_libc_bool_opr_rec(sou->when);
    tar->when_start = (struct libc_bool_opr_rec *)
copy_libc_bool_opr_rec(sou->when_start);
    tar->when_end   = (struct libc_bool_opr_rec *)
copy_libc_bool_opr_rec(sou->when_end);
    tar->piecewise  = (struct libc_piece_value_rec *)
copy_libc_piece_value_rec(sou->piecewise);
    tar->cell_rise  = (struct libc_table_val_rec *)
copy_libc_table_val_rec(sou->cell_rise);
    tar->cell_fall  = (struct libc_table_val_rec *)
copy_libc_table_val_rec(sou->cell_fall);
    tar->rise_propagation = (struct libc_table_val_rec *)
copy_libc_table_val_rec(sou->rise_propagation);
    tar->fall_propagation = (struct libc_table_val_rec *)
copy_libc_table_val_rec(sou->fall_propagation);
    tar->rise_transition = (struct libc_table_val_rec *)
copy_libc_table_val_rec(sou->rise_transition);
```

```
    tar->fall_transition = (struct libc_table_val_rec *)
copy_libc_table_val_rec(sou->fall_transition);
    tar->rise_constraint = (struct libc_table_val_rec *)
copy_libc_table_val_rec(sou->rise_constraint);
    tar->fall_constraint = (struct libc_table_val_rec *)
copy_libc_table_val_rec(sou->fall_constraint);
    sou  = sou->next ;
    if (sou!=NULL)
    { prev = tar;
      tar->next = new_libc_timing_rec();
      tar = tar->next;
    }
  }
    return(head);
}


/* ========================================================
   routine:  free_libc_min_pulse_width_rec()
   ======================================================== */

void  free_libc_min_pulse_width_rec(struct libc_min_pulse_width_rec *ptr)
{
  struct libc_min_pulse_width_rec *next;

  while (ptr!=NULL) {
    free_libc_bool_opr_rec(ptr->when);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,3);
#if DMP_MEM_ANY
    dmp_libc_min_pulse_width_rec_msize_dec(16);
    dmp_libc_mem_size_dec(16);
#endif
    ptr = next ;
  }
}

/* ========================================================
   routine: nfree_libc_min_pulse_width_rec()
   ======================================================== */

void nfree_libc_min_pulse_width_rec(struct libc_min_pulse_width_rec **pptr)
{ struct libc_min_pulse_width_rec *ptr= *pptr;

  struct libc_min_pulse_width_rec *next;

  while (ptr!=NULL) {
    free_libc_bool_opr_rec(ptr->when);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,3);
#if DMP_MEM_ANY
    dmp_libc_min_pulse_width_rec_msize_dec(16);
    dmp_libc_mem_size_dec(16);
#endif
    ptr = next ;
  }
  *pptr = NULL;
```

```
}

struct libc_min_pulse_width_rec * copy_libc_min_pulse_width_rec(struct
libc_min_pulse_width_rec *sou)
{ struct libc_min_pulse_width_rec *tar,*head,*prev;

  if (sou==NULL) return(NULL);
  prev = NULL;
  head = tar = new_libc_min_pulse_width_rec();
  while (sou!=NULL) {
    tar->constraint_high = sou->constraint_high;
    tar->constraint_low = sou->constraint_low;
    tar->when       = (struct libc_bool_opr_rec *)
copy_libc_bool_opr_rec(sou->when);
    sou  = sou->next ;
    if (sou!=NULL)
    { prev = tar;
      tar->next = new_libc_min_pulse_width_rec();
      tar = tar->next;
    }
  }
    return(head);
}


/* ========================================================
   routine:  free_libc_minimum_period_rec()
   ======================================================== */

void  free_libc_minimum_period_rec(struct libc_minimum_period_rec *ptr)
{
  struct libc_minimum_period_rec *next;

  while (ptr!=NULL) {
    free_libc_bool_opr_rec(ptr-> when);
    next = ptr->next ;
    free_mb_ptr((void *) ptr,2);
#if DMP_MEM_ANY
    dmp_libc_minimum_period_rec_msize_dec(12);
    dmp_libc_mem_size_dec(12);
#endif
    ptr = next ;
  }
}

/* ========================================================
   routine: nfree_libc_minimum_period_rec()
   ======================================================== */

void nfree_libc_minimum_period_rec(struct libc_minimum_period_rec **pptr)
{ struct libc_minimum_period_rec *ptr= *pptr;

  struct libc_minimum_period_rec *next;

  while (ptr!=NULL) {
    free_libc_bool_opr_rec(ptr->when);
     next = ptr->next ;
```

```
    free_mb_ptr((void *) ptr,2);
#if DMP_MEM_ANY
    dmp_libc_minimum_period_rec_msize_dec(12);
    dmp_libc_mem_size_dec(12);
#endif
    ptr = next ;
  }
  *pptr = NULL;
}

struct libc_minimum_period_rec * copy_libc_minimum_period_rec(struct
libc_minimum_period_rec *sou)
{ struct libc_minimum_period_rec *tar,*head,*prev;

  if (sou==NULL) return(NULL);
  prev = NULL;
  head = tar = new_libc_minimum_period_rec();
  while (sou!=NULL) {
    tar->constraint = sou->constraint;
    tar->when       = (struct libc_bool_opr_rec *)
copy_libc_bool_opr_rec(sou->when);
    sou  = sou->next ;
    if (sou!=NULL)
    { prev = tar;
      tar->next = new_libc_minimum_period_rec();
      tar = tar->next;
    }
  }
    return(head);
}


/* ========================================================
   routine:  free_libc_routing_track_rec()
   ======================================================== */

void  free_libc_routing_track_rec(struct libc_routing_track_rec *ptr)
{
  struct libc_routing_track_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr->layer_name);
    next = ptr->next ;
    free_mb_ptr((void *) ptr,3);
#if DMP_MEM_ANY
    dmp_libc_routing_track_rec_msize_dec(16);
    dmp_libc_mem_size_dec(16);
#endif
    ptr = next ;
  }
}

/* ========================================================
   routine: nfree_libc_routing_track_rec()
   ======================================================== */

void nfree_libc_routing_track_rec(struct libc_routing_track_rec **pptr)
```

```
{ struct libc_routing_track_rec *ptr= *pptr;

  struct libc_routing_track_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr->layer_name);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,3);
#if DMP_MEM_ANY
    dmp_libc_routing_track_rec_msize_dec(16);
    dmp_libc_mem_size_dec(16);
#endif
    ptr = next ;
  }
  *pptr = NULL;
}

/* ============================================================
   routine:  free_libc_def_entry_rec()
   ============================================================ */

void  free_libc_def_entry_rec(struct libc_def_entry_rec *ptr)
{
  struct libc_def_entry_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr-> def_name);
    next = ptr->next ;
    free_mb_ptr((void *) ptr,2);
#if DMP_MEM_ANY
    dmp_libc_def_entry_rec_msize_dec(12);
    dmp_libc_mem_size_dec(12);
#endif
    ptr = next ;
  }
}

/* ============================================================
   routine: nfree_libc_def_entry_rec()
   ============================================================ */

void nfree_libc_def_entry_rec(struct libc_def_entry_rec **pptr)
{ struct libc_def_entry_rec *ptr= *pptr;

  struct libc_def_entry_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr-> def_name);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,2);
#if DMP_MEM_ANY
    dmp_libc_def_entry_rec_msize_dec(12);
    dmp_libc_mem_size_dec(12);
#endif
    ptr = next ;
  }
  *pptr = NULL;
```

```
}

/* ======================================================
   routine:  free_libc_define_value_rec()
   ====================================================== */

void  free_libc_define_value_rec(struct libc_define_value_rec *ptr)
{
  struct libc_define_value_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr->str_val);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,3);
#if DMP_MEM_ANY
    dmp_libc_define_value_rec_msize_dec(16);
    dmp_libc_mem_size_dec(16);
#endif
    ptr = next ;
  }
}

/* ======================================================
   routine: nfree_libc_define_value_rec()
   ====================================================== */

void nfree_libc_define_value_rec(struct libc_define_value_rec **pptr)
{ struct libc_define_value_rec *ptr= *pptr;

  struct libc_define_value_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr->str_val);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,3);
#if DMP_MEM_ANY
    dmp_libc_define_value_rec_msize_dec(16);
    dmp_libc_mem_size_dec(16);
#endif
    ptr = next ;
  }
  *pptr = NULL;
}

struct libc_define_value_rec * copy_libc_define_value_rec(struct
libc_define_value_rec *sou)
{ struct libc_define_value_rec *tar,*head,*prev;

  if (sou==NULL) return(NULL);
  prev = NULL;
  head = tar = new_libc_define_value_rec();
  while (sou!=NULL) {
    tar->_def       = sou->_def;
    tar->float_val  = sou->float_val;
    tar->str_val    = (char *) copy_string(sou->str_val);
    sou  = sou->next ;
    if (sou!=NULL)
```

```
    { prev = tar;
      tar->next = new_libc_define_value_rec();
      tar = tar->next;
    }
  }
    return(head);
}


/* ========================================================
   routine:  free_libc_glb_const_rec()
   ======================================================== */

void  free_libc_glb_const_rec(struct libc_glb_const_rec *ptr)
{
  struct libc_glb_const_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr->gc_name);
     next = ptr->next ;
    free_mb_ptr((void *) ptr,2);
#if DMP_MEM_ANY
    dmp_libc_glb_const_rec_msize_dec(12);
    dmp_libc_mem_size_dec(12);
#endif
    ptr = next ;
  }
}

/* ========================================================
   routine: nfree_libc_glb_const_rec()
   ======================================================== */

void nfree_libc_glb_const_rec(struct libc_glb_const_rec **pptr)
{ struct libc_glb_const_rec *ptr= *pptr;

  struct libc_glb_const_rec *next;

  while (ptr!=NULL) {
    free_text_buffer(ptr-> gc_name);
    next = ptr->next ;
    free_mb_ptr((void *) ptr,2);
#if DMP_MEM_ANY
    dmp_libc_glb_const_rec_msize_dec(12);
    dmp_libc_mem_size_dec(12);
#endif
    ptr = next ;
  }
  *pptr = NULL;
}

/* ========================================================
   routine:  free_libc_def_table_rec()
   ======================================================== */

void  free_libc_def_table_rec(struct libc_def_table_rec *ptr)
{ int i0;
```

```
  if (ptr==NULL) return;
    if (ptr->gc_entry!=NULL) {
    for (i0=0;i0<sizeof_ptr_buffer(ptr->gc_entry);i0++)
       free_libc_glb_const_rec(ptr->gc_entry[i0]);
    free_ptr_buffer(ptr->gc_entry);
    }
    if (ptr->lib!=NULL) {
    for (i0=0;i0<sizeof_ptr_buffer(ptr->lib);i0++)
       free_libc_def_entry_rec(ptr->lib[i0]);
    free_ptr_buffer(ptr->lib);
    }
    if (ptr-> cell!=NULL) {
    for (i0=0;i0<sizeof_ptr_buffer(ptr-> cell);i0++)
       free_libc_def_entry_rec(ptr-> cell[i0]);
    free_ptr_buffer(ptr-> cell);
    }
    if (ptr->pin!=NULL) {
    for (i0=0;i0<sizeof_ptr_buffer(ptr->pin);i0++)
       free_libc_def_entry_rec(ptr->pin[i0]);
    free_ptr_buffer(ptr->pin);
    }
    free_mb_ptr((void *) ptr,3);
#if DMP_MEM_ANY
    dmp_libc_def_table_rec_msize_dec(16);
    dmp_libc_mem_size_dec(16);
#endif
}

/* ============================================================
   routine: nfree_libc_def_table_rec()
   ============================================================ */

void nfree_libc_def_table_rec(struct libc_def_table_rec **pptr)
{ struct libc_def_table_rec *ptr= *pptr;
 int i0;
  if (ptr==NULL) return;
    if (ptr->gc_entry!=NULL) {
    for (i0=0;i0<sizeof_ptr_buffer(ptr->gc_entry);i0++)
       free_libc_glb_const_rec(ptr->gc_entry[i0]);
    free_ptr_buffer(ptr->gc_entry);
    }
    if (ptr-> lib!=NULL) {
    for (i0=0;i0<sizeof_ptr_buffer(ptr-> lib);i0++)
       free_libc_def_entry_rec(ptr-> lib[i0]);
    free_ptr_buffer(ptr-> lib);
    }
    if (ptr-> cell!=NULL) {
    for (i0=0;i0<sizeof_ptr_buffer(ptr-> cell);i0++)
       free_libc_def_entry_rec(ptr-> cell[i0]);
    free_ptr_buffer(ptr-> cell);
    }
    if (ptr-> pin!=NULL) {
    for (i0=0;i0<sizeof_ptr_buffer(ptr-> pin);i0++)
       free_libc_def_entry_rec(ptr-> pin[i0]);
    free_ptr_buffer(ptr-> pin);
    }
    free_mb_ptr((void *) ptr,3);
```

```
#if DMP_MEM_ANY
    dmp_libc_def_table_rec_msize_dec(16);
    dmp_libc_mem_size_dec(16);
#endif
  *pptr = NULL;
}
```

```
#ifndef _H_libc_mem_DEF
#define _H_libc_mem_DEF 1
#include <stdio.h>
#include "dmp_util.h"
#include "libcds.h"
#if DMP_MEM_ANY
struct any_unit_rec *new_any_unit_rec(void);
struct libc_name_list_rec *new_libc_name_list_rec(void);
struct libc_name_list_list *new_libc_name_list_list(void);
struct libc_float_list_rec *new_libc_float_list_rec(void);
struct libc_float_list_list *new_libc_float_list_list(void);
struct libc_define_rec *new_libc_define_rec(void);
struct libc_cell_area_rec *new_libc_cell_area_rec(void);
struct libc_k_factor_rec *new_libc_k_factor_rec(void);
struct libc_lib_rec *new_libc_lib_rec(void);
struct libc_lu_table_template_rec *new_libc_lu_table_template_rec(void);
struct libc_oc_power_rail_rec *new_libc_oc_power_rail_rec(void);
struct libc_operating_condition_rec *new_libc_operating_condition_rec(void);
struct libc_power_supply_rec *new_libc_power_supply_rec(void);
struct libc_timing_range_rec *new_libc_timing_range_rec(void);
struct libc_type_rec *new_libc_type_rec(void);
struct libc_fanout_length_rec *new_libc_fanout_length_rec(void);
struct libc_wire_load_rec *new_libc_wire_load_rec(void);
struct libc_wire_load_from_area_rec *new_libc_wire_load_from_area_rec(void);
struct libc_wire_load_selection_rec *new_libc_wire_load_selection_rec(void);
struct libc_cell_rec *new_libc_cell_rec(void);
struct libc_memory_write_rec *new_libc_memory_write_rec(void);
struct libc_bool_opr_rec *new_libc_bool_opr_rec(void);
struct libc_pin_rec *new_libc_pin_rec(void);
struct libc_ff_latch_rec *new_libc_ff_latch_rec(void);
struct libc_internal_power *new_libc_internal_power(void);
struct libc_leakage_power *new_libc_leakage_power(void);
struct libc_memory_rec *new_libc_memory_rec(void);
struct libc_piece_value_rec *new_libc_piece_value_rec(void);
struct libc_float_rec *new_libc_float_rec(void);
struct libc_table_val_rec *new_libc_table_val_rec(void);
struct libc_timing_rec *new_libc_timing_rec(void);
struct libc_min_pulse_width_rec *new_libc_min_pulse_width_rec(void);
struct libc_minimum_period_rec *new_libc_minimum_period_rec(void);
struct libc_routing_track_rec *new_libc_routing_track_rec(void);
struct libc_def_entry_rec *new_libc_def_entry_rec(void);
struct libc_define_value_rec *new_libc_define_value_rec(void);
struct libc_glb_const_rec *new_libc_glb_const_rec(void);
struct libc_def_table_rec *new_libc_def_table_rec(void);
#else
#define new_any_unit_rec() (struct any_unit_rec *) get_mb_ptr(1)
#define new_libc_name_list_rec() (struct libc_name_list_rec *) get_mb_ptr(1)
#define new_libc_name_list_list() (struct libc_name_list_list *)
get_mb_ptr(2)
#define new_libc_float_list_rec() (struct libc_float_list_rec *)
get_mb_ptr(1)
#define new_libc_float_list_list() (struct libc_float_list_list *)
get_mb_ptr(1)
#define new_libc_define_rec() (struct libc_define_rec *) get_mb_ptr(3)
#define new_libc_cell_area_rec() (struct libc_cell_area_rec *) get_mb_ptr(2)
#define new_libc_k_factor_rec() (struct libc_k_factor_rec *) get_mb_ptr(29)
#define new_libc_lib_rec() (struct libc_lib_rec *) get_mb_ptr(28)
```

```
#define new_libc_lu_table_template_rec() (struct libc_lu_table_template_rec
*) get_mb_ptr(7)
#define new_libc_oc_power_rail_rec() (struct libc_oc_power_rail_rec *)
get_mb_ptr(2)
#define new_libc_operating_condition_rec() (struct
libc_operating_condition_rec *) get_mb_ptr(6)
#define new_libc_power_supply_rec() (struct libc_power_supply_rec *)
get_mb_ptr(3)
#define new_libc_timing_range_rec() (struct libc_timing_range_rec *)
get_mb_ptr(3)
#define new_libc_type_rec() (struct libc_type_rec *) get_mb_ptr(5)
#define new_libc_fanout_length_rec() (struct libc_fanout_length_rec *)
get_mb_ptr(8)
#define new_libc_wire_load_rec() (struct libc_wire_load_rec *) get_mb_ptr(6)
#define new_libc_wire_load_from_area_rec() (struct
libc_wire_load_from_area_rec *) get_mb_ptr(3)
#define new_libc_wire_load_selection_rec() (struct
libc_wire_load_selection_rec *) get_mb_ptr(2)
#define new_libc_cell_rec() (struct libc_cell_rec *) get_mb_ptr(23)
#define new_libc_memory_write_rec() (struct libc_memory_write_rec *)
get_mb_ptr(2)
#define new_libc_bool_opr_rec() (struct libc_bool_opr_rec *) get_mb_ptr(3)
#define new_libc_pin_rec() (struct libc_pin_rec *) get_mb_ptr(27)
#define new_libc_ff_latch_rec() (struct libc_ff_latch_rec *) get_mb_ptr(15)
#define new_libc_internal_power() (struct libc_internal_power *)
get_mb_ptr(9)
#define new_libc_leakage_power() (struct libc_leakage_power *) get_mb_ptr(2)
#define new_libc_memory_rec() (struct libc_memory_rec *) get_mb_ptr(2)
#define new_libc_piece_value_rec() (struct libc_piece_value_rec *)
get_mb_ptr(9)
#define new_libc_float_rec() (struct libc_float_rec *) get_mb_ptr(0)
#define new_libc_table_val_rec() (struct libc_table_val_rec *) get_mb_ptr(5)
#define new_libc_timing_rec() (struct libc_timing_rec *) get_mb_ptr(22)
#define new_libc_min_pulse_width_rec() (struct libc_min_pulse_width_rec *)
get_mb_ptr(3)
#define new_libc_minimum_period_rec() (struct libc_minimum_period_rec *)
get_mb_ptr(2)
#define new_libc_routing_track_rec() (struct libc_routing_track_rec *)
get_mb_ptr(3)
#define new_libc_def_entry_rec() (struct libc_def_entry_rec *) get_mb_ptr(2)
#define new_libc_define_value_rec() (struct libc_define_value_rec *)
get_mb_ptr(3)
#define new_libc_glb_const_rec() (struct libc_glb_const_rec *) get_mb_ptr(2)
#define new_libc_def_table_rec() (struct libc_def_table_rec *) get_mb_ptr(3)
#endif
void  free_any_unit_rec(struct any_unit_rec *);
void nfree_any_unit_rec(struct any_unit_rec **);
void  free_libc_name_list_rec(struct libc_name_list_rec *);
void nfree_libc_name_list_rec(struct libc_name_list_rec **);
struct libc_name_list_rec * copy_libc_name_list_rec(struct libc_name_list_rec
*);
void  free_libc_name_list_list(struct libc_name_list_list *);
void nfree_libc_name_list_list(struct libc_name_list_list **);
void  free_libc_float_list_rec(struct libc_float_list_rec *);
void nfree_libc_float_list_rec(struct libc_float_list_rec **);
void  free_libc_float_list_list(struct libc_float_list_list *);
void nfree_libc_float_list_list(struct libc_float_list_list **);
```

```
void  free_libc_define_rec(struct libc_define_rec *);
void nfree_libc_define_rec(struct libc_define_rec **);
void  free_libc_cell_area_rec(struct libc_cell_area_rec *);
void nfree_libc_cell_area_rec(struct libc_cell_area_rec **);
void  free_libc_k_factor_rec(struct libc_k_factor_rec *);
void nfree_libc_k_factor_rec(struct libc_k_factor_rec **);
void  free_libc_lib_rec(struct libc_lib_rec *);
void nfree_libc_lib_rec(struct libc_lib_rec **);
void  free_libc_lu_table_template_rec(struct libc_lu_table_template_rec *);
void nfree_libc_lu_table_template_rec(struct libc_lu_table_template_rec **);
void  free_libc_oc_power_rail_rec(struct libc_oc_power_rail_rec *);
void nfree_libc_oc_power_rail_rec(struct libc_oc_power_rail_rec **);
void  free_libc_operating_condition_rec(struct libc_operating_condition_rec
*);
void nfree_libc_operating_condition_rec(struct libc_operating_condition_rec
**);
void  free_libc_power_supply_rec(struct libc_power_supply_rec *);
void nfree_libc_power_supply_rec(struct libc_power_supply_rec **);
void  free_libc_timing_range_rec(struct libc_timing_range_rec *);
void nfree_libc_timing_range_rec(struct libc_timing_range_rec **);
void  free_libc_type_rec(struct libc_type_rec *);
void nfree_libc_type_rec(struct libc_type_rec **);
void  free_libc_fanout_length_rec(struct libc_fanout_length_rec *);
void nfree_libc_fanout_length_rec(struct libc_fanout_length_rec **);
void  free_libc_wire_load_rec(struct libc_wire_load_rec *);
void nfree_libc_wire_load_rec(struct libc_wire_load_rec **);
void  free_libc_wire_load_from_area_rec(struct libc_wire_load_from_area_rec
*);
void nfree_libc_wire_load_from_area_rec(struct libc_wire_load_from_area_rec
**);
void  free_libc_wire_load_selection_rec(struct libc_wire_load_selection_rec
*);
void nfree_libc_wire_load_selection_rec(struct libc_wire_load_selection_rec
**);
void  free_libc_cell_rec(struct libc_cell_rec *);
void nfree_libc_cell_rec(struct libc_cell_rec **);
void  free_libc_memory_write_rec(struct libc_memory_write_rec *);
void nfree_libc_memory_write_rec(struct libc_memory_write_rec **);
struct libc_memory_write_rec * copy_libc_memory_write_rec(struct
libc_memory_write_rec *);
void  free_libc_bool_opr_rec(struct libc_bool_opr_rec *);
void nfree_libc_bool_opr_rec(struct libc_bool_opr_rec **);
struct libc_bool_opr_rec * copy_libc_bool_opr_rec(struct libc_bool_opr_rec
*);
void  free_libc_pin_rec(struct libc_pin_rec *);
void nfree_libc_pin_rec(struct libc_pin_rec **);
struct libc_pin_rec * copy_libc_pin_rec(struct libc_pin_rec *);
void  free_libc_ff_latch_rec(struct libc_ff_latch_rec *);
void nfree_libc_ff_latch_rec(struct libc_ff_latch_rec **);
void  free_libc_internal_power(struct libc_internal_power *);
void nfree_libc_internal_power(struct libc_internal_power **);
struct libc_internal_power * copy_libc_internal_power(struct
libc_internal_power *);
void  free_libc_leakage_power(struct libc_leakage_power *);
void nfree_libc_leakage_power(struct libc_leakage_power **);
struct libc_leakage_power * copy_libc_leakage_power(struct libc_leakage_power
*);
```

```
void  free_libc_memory_rec(struct libc_memory_rec *);
void nfree_libc_memory_rec(struct libc_memory_rec **);
void  free_libc_piece_value_rec(struct libc_piece_value_rec *);
void nfree_libc_piece_value_rec(struct libc_piece_value_rec **);
struct libc_piece_value_rec * copy_libc_piece_value_rec(struct
libc_piece_value_rec *);
void  free_libc_float_rec(struct libc_float_rec *);
void nfree_libc_float_rec(struct libc_float_rec **);
struct libc_float_rec * copy_libc_float_rec(struct libc_float_rec *);
void  free_libc_table_val_rec(struct libc_table_val_rec *);
void nfree_libc_table_val_rec(struct libc_table_val_rec **);
struct libc_table_val_rec * copy_libc_table_val_rec(struct libc_table_val_rec
*);
void  free_libc_timing_rec(struct libc_timing_rec *);
void nfree_libc_timing_rec(struct libc_timing_rec **);
struct libc_timing_rec * copy_libc_timing_rec(struct libc_timing_rec *);
void  free_libc_min_pulse_width_rec(struct libc_min_pulse_width_rec *);
void nfree_libc_min_pulse_width_rec(struct libc_min_pulse_width_rec **);
struct libc_min_pulse_width_rec * copy_libc_min_pulse_width_rec(struct
libc_min_pulse_width_rec *);
void  free_libc_minimum_period_rec(struct libc_minimum_period_rec *);
void nfree_libc_minimum_period_rec(struct libc_minimum_period_rec **);
struct libc_minimum_period_rec * copy_libc_minimum_period_rec(struct
libc_minimum_period_rec *);
void  free_libc_routing_track_rec(struct libc_routing_track_rec *);
void nfree_libc_routing_track_rec(struct libc_routing_track_rec **);
void  free_libc_def_entry_rec(struct libc_def_entry_rec *);
void nfree_libc_def_entry_rec(struct libc_def_entry_rec **);
void  free_libc_define_value_rec(struct libc_define_value_rec *);
void nfree_libc_define_value_rec(struct libc_define_value_rec **);
struct libc_define_value_rec * copy_libc_define_value_rec(struct
libc_define_value_rec *);
void  free_libc_glb_const_rec(struct libc_glb_const_rec *);
void nfree_libc_glb_const_rec(struct libc_glb_const_rec **);
void  free_libc_def_table_rec(struct libc_def_table_rec *);
void nfree_libc_def_table_rec(struct libc_def_table_rec **);
#endif
```

```
/*================================================


 ================================================ */

#include "libc_def.h"

/* ================================================ */

public
libc_bool_opr_rec *libc_opr_handle(
      libc_bool_type_E op_type,
      int value)
{ libc_bool_opr_rec * bor;

  bor = new_libc_bool_opr_rec();
  bor->type = op_type;
  if (op_type == ID_B)
    bor->u.id_name = (text_buffer *) value;
  else
    bor->u.value = value;
  return(bor);
}

/* ================================================ */

public
void libc_opr_id_print(
      FILE *outp,
      char *name,
      libc_cell_rec *cell,
      int idx)
{ libc_pin_rec *pin;
  libc_ff_latch_rec *ffp;

  pin = libc_cell_find_pin_by_name(cell,name,-99999);
  if (pin == NULL) {          /* (idx >= 0) and (this ID_B is not a bundle or
bus) */
    /* ---- for index of ff_latch (IQ,IQN) */
    if (idx > -10000) {
#if 1
      fprintf(outp,cell->_tlib->bus_naming_style,name,idx);
#else
      for (ffp=cell->ff_latch;ffp;ffp=ffp->next) {
        if (strcmp(cell->ff_latch->Q_name,name)==0 ||
          strcmp(cell->ff_latch->QN_name,name)==0) {
          fprintf(outp,cell->_tlib->bus_naming_style,name,idx);
        return;
      }
      }
      fprintf(outp,"%s",name);
#endif
    }
    else
      fprintf(outp,"%s",name);
  }
```

```
  else {
    if (pin->is_bus)
      fprintf(outp,cell->_tlib->bus_naming_style,name,idx);
    else if (pin->members != NULL) {
      libc_name_list_rec *np;
      int i;

      for (np=pin->members,i=0;np!=NULL;np=np->next,i++) {
      if (i == idx)
        break;
      }
      assert(np != NULL);
      fprintf(outp,"%s",np->name);
    }
    else
      fprintf(outp,"%s",name);
  }
}

/* ------------------------------------------------ */

public
void libc_opr_print(
      FILE *outp,
      libc_bool_opr_rec *opr,
      libc_cell_rec *cell,
      int idx)
{ libc_pin_rec *pin;

  if (opr == NULL)
    return;
  switch (opr->type) {
    case XOR_B :
    case OR_B :
    case AND_B :
      if (opr->L->type >= opr->type)
      libc_opr_print(outp,opr->L,cell,idx);
      else {
        fprintf(outp,"(");
      libc_opr_print(outp,opr->L,cell,idx);
        fprintf(outp,")");
      }
      fprintf(outp,"%s",(opr->type == AND_B)? " " : (opr->type == OR_B)? " |
" : " ^ ");
      if (opr->R->type >= opr->type)
      libc_opr_print(outp,opr->R,cell,idx);
      else {
        fprintf(outp,"(");
      libc_opr_print(outp,opr->R,cell,idx);
        fprintf(outp,")");
      }
      break;
    case NOT_B :
      fprintf(outp,"!");
      if (opr->R->type >= NOT_B)
      libc_opr_print(outp,opr->R,cell,idx);
      else {
```

```
        fprintf(outp,"(");
      libc_opr_print(outp,opr->R,cell,idx);
        fprintf(outp,")");
      }
      break;
    case ID_B :
      libc_opr_id_print(outp,opr->u.id_name,cell,idx);
      break;
    case INDEX_B :
      libc_opr_print(outp,opr->L,cell,opr->R->u.value);
      break;
    case ZERO_B :
    case ONE_B :
    case CONST_B :
      fprintf(outp,"%d",opr->u.value);
      break;
  }
}

/* ================================================ */

public
void libc_opr_power_when_print(
      FILE *outp,
      libc_bool_opr_rec *opr,
      libc_cell_rec *cell,
      int idx)
{ libc_pin_rec *pin;

  if (opr == NULL)
    return;
  switch (opr->type) {
    case XOR_B :
    case OR_B :
    case AND_B :
      if (opr->L->type >= opr->type)
      libc_opr_power_when_print(outp,opr->L,cell,idx);
      else {
        fprintf(outp,"(");
      libc_opr_power_when_print(outp,opr->L,cell,idx);
        fprintf(outp,")");
      }
      fprintf(outp,"%s",(opr->type == AND_B)? "*" : (opr->type == OR_B)? "+"
: "^");
      if (opr->R->type >= opr->type)
      libc_opr_power_when_print(outp,opr->R,cell,idx);
      else {
        fprintf(outp,"(");
      libc_opr_power_when_print(outp,opr->R,cell,idx);
        fprintf(outp,")");
      }
      break;
    case NOT_B :
      if (opr->R->type == ID_B) {
        libc_opr_id_print(outp,opr->R->u.id_name,cell,idx);
        fprintf(outp,"0");
      }
```

```
      else {
        fprintf(outp,"!");
        if (opr->R->type >= NOT_B)
        libc_opr_power_when_print(outp,opr->R,cell,idx);
        else {
          fprintf(outp,"(");
        libc_opr_power_when_print(outp,opr->R,cell,idx);
          fprintf(outp,")");
        }
      }
      break;
    case ID_B :
      libc_opr_id_print(outp,opr->u.id_name,cell,idx);
      fprintf(outp,"1");
      break;
    case INDEX_B :
      libc_opr_power_when_print(outp,opr->L,cell,opr->R->u.value);
      break;
    case ZERO_B :
    case ONE_B :
    case CONST_B :
      fprintf(outp,"%d",opr->u.value);
      break;
  }
}

/* ------------------------------------------------- */

/* ================================================= */
```

```
/*================================================

 ================================================ */

#include "libc_def.h"

/* ================================================ */

  static
  char *libc_time_delay_str(
        delay_model_E de)
  {
    switch (de) {
      case GENERIC_ECL :       return("generic_ecl");
      case GENERIC_CMOS :      return("generic_cmos");
      case TABLE_LOOKUP :      return("table_lookup");
      case CMOS2 :             return("cmos2");
      case PIECEWISE_CMOS :    return("piecewise_cmos");
      deafult :
      assert(0);
    }
  }

/* ------------------------------------------------ */

public
void libc_time_delay_model(
      delay_model_E de)
{ char msg[256];

  if (tech_lib->delay_model == UNKNOW_M) {
    if (de != GENERIC_CMOS && de != TABLE_LOOKUP) {
      sprintf(msg,"This statement belongs to %s delay model,\n
only generic_cmos and table_lookup delay model are supported.",
      libc_time_delay_str(de));
      libcerror(msg);
    }
    tech_lib->delay_model = de;
  }
  else if (tech_lib->delay_model == de)
    return ;
  else {
    sprintf(msg,"\"%s\" statement in \"%s\" delay model.",
      libc_time_delay_str(de),libc_time_delay_str(tech_lib->delay_model));
    libcerror(msg);
    if (tech_lib->delay_model != GENERIC_CMOS &&
        tech_lib->delay_model != TABLE_LOOKUP)
      tech_lib->delay_model = de;
  }
}

/* ================================================ */

public
void libc_time_init(
```

```
      text_buffer *timing_name)
{
  lib_timing = new_libc_timing_rec();
  lib_timing->t_name = timing_name;
  lib_timing->_current_pin = lib_pin;

  /* ---- set default value */
  lib_timing->intrinsic_fall  = tech_lib->default_intrinsic_fall;
  lib_timing->intrinsic_rise  = tech_lib->default_intrinsic_rise;
  lib_timing->slope_fall      = tech_lib->default_slope_fall;
  lib_timing->slope_rise      = tech_lib->default_slope_rise;
  if (lib_pin->direction == INOUT_E) {
    lib_timing->fall_resistance     = tech_lib->default_inout_pin_fall_res;
    lib_timing->rise_resistance     = tech_lib->default_inout_pin_rise_res;
  }
  else {
    lib_timing->fall_resistance     = tech_lib->default_output_pin_fall_res;
    lib_timing->rise_resistance     = tech_lib->default_output_pin_rise_res;
  }
}

/* ------------------------------------------------ */

public
void libc_time_finish(void)
{ char msg[128],*type_name;
  int t_type;

  switch (lib_timing->timing_type) {
    case COMBINATIONAL_T :     t_type = 1; type_name = "combination";
      break;
    case RISING_EDGE_T :       t_type = 1; type_name = "rising_edge";
      break;
    case FALLING_EDGE_T :      t_type = 1; type_name = "falling_edge";
      break;
    case PRESET_T :            t_type = 3; type_name = "preset";
      break;
    case CLEAR_T :             t_type = 3; type_name = "clear";
      break;
    case THREE_STATE_DISABLE_T :t_type = 1;     type_name =
"three_state_disable";  break;
    case THREE_STATE_ENABLE_T :     t_type = 1; type_name =
"three_state_enable";   break;

    case HOLD_RISING_T :       t_type = 4; type_name = "hold_rising";
      break;
    case HOLD_FALLING_T :      t_type = 4; type_name = "hold_falling";
      break;
    case SETUP_RISING_T :      t_type = 4; type_name = "setup_rising";
      break;
    case SETUP_FALLING_T :     t_type = 4; type_name = "setup_falling";
      break;
    case RECOVERY_RISING_T :   t_type = 4; type_name = "recovery_rising";
      break;
    case RECOVERY_FALLING_T : t_type = 4; type_name = "recovery_falling";
      break;
```

```
    case REMOVAL_RISING_T :   t_type = 4; type_name = "removal_rising";
      break;
    case REMOVAL_FALLING_T :  t_type = 4; type_name = "removal_falling";
      break;

    case SKEW_RISING_T :      t_type = 2; type_name = "skew_rising";
      break;
    case SKEW_FALLING_T :     t_type = 2; type_name = "skew_falling";
      break;
    case NON_SEQ_HOLD_RISING_T :   t_type = 2;  type_name =
"non_seq_hold_rising";  break;
    case NON_SEQ_HOLD_FALLING_T :  t_type = 2;  type_name =
"non_seq_hold_falling"; break;
    case NON_SEQ_SETUP_RISING_T :  t_type = 2;  type_name =
"non_seq_setup_rising"; break;
    case NON_SEQ_SETUP_FALLING_T : t_type = 2;  type_name =
"non_seq_setup_falling";      break;
    case NOCHANGE_HIGH_HIGH_T :     t_type = 2; type_name =
"nochange_high_high";   break;
    case NOCHANGE_HIGH_LOW_T :      t_type = 2; type_name =
"nochange_high_low";    break;
    case NOCHANGE_LOW_HIGH_T :      t_type = 2; type_name =
"nochange_low_high";    break;
    case NOCHANGE_LOW_LOW_T : t_type = 2; type_name = "nochange_low_low";
      break;

    default :
      libcerror("timing_type in timing group is not specified.");
      return;
  }

  if (tech_lib->delay_model != TABLE_LOOKUP)
    return;

  if (t_type == 1) {
    if (lib_timing->cell_rise == NULL && lib_timing->rise_propagation ==
NULL) {
      sprintf(msg,"cell_rise or rise_propagation table is required for
timing_type %s.",type_name);
      libcerror(msg);
    }
    else if (lib_timing->cell_rise != NULL && lib_timing->rise_propagation !=
NULL) {
      sprintf(msg,"only one table is allowed of cell_rise or rise_propagation
tables for timing_type %s.",type_name);
      libcerror(msg);
    }
    if (lib_timing->cell_fall == NULL && lib_timing->fall_propagation ==
NULL) {
      sprintf(msg,"cell_fall or fall_propagation table is required for
timing_type %s.",type_name);
      libcerror(msg);
    }
    else if (lib_timing->cell_fall == NULL && lib_timing->fall_propagation ==
NULL) {
      sprintf(msg,"only one table is allowed of cell_fall or fall_propagation
tables for timing_type %s.",type_name);
```

```
      libcerror(msg);
    }
    else if (lib_timing->cell_rise != NULL && lib_timing->cell_fall == NULL)
{
      sprintf(msg,"cell_fall table is required for timing_type
%s.",type_name);
      libcerror(msg);
    }
    else if (lib_timing->rise_propagation != NULL && lib_timing-
>fall_propagation == NULL) {
      sprintf(msg,"fall_propagation table is required for timing_type
%s.",type_name);
      libcerror(msg);
    }

    if (lib_timing->rise_transition == NULL) {
      sprintf(msg,"rise_transition table is required for timing_type
%s.",type_name);
      libcerror(msg);
    }
    if (lib_timing->fall_transition == NULL) {
      sprintf(msg,"fall_transition table is required for timing_type
%s.",type_name);
      libcerror(msg);
    }
  }
  else if (t_type == 2) {
    if (lib_timing->rise_constraint == NULL) {
      sprintf(msg,"rise_constraint table is required for timing_type
%s.",type_name);
      libcerror(msg);
    }
    if (lib_timing->fall_constraint == NULL) {
      sprintf(msg,"fall_constraint table is required for timing_type
%s.",type_name);
      libcerror(msg);
    }
  }
  else if (t_type == 3) {
    if (lib_timing->cell_rise == NULL && lib_timing->rise_propagation == NULL
&&
        lib_timing->cell_fall == NULL && lib_timing->fall_propagation ==
NULL) {
      sprintf(msg,"delay or propagation table is required for timing_type
%s.",type_name);
      libcerror(msg);
    }
    if (lib_timing->rise_transition == NULL && lib_timing->fall_transition ==
NULL) {
      sprintf(msg,"transition table is required for timing_type
%s.",type_name);
      libcerror(msg);
    }
  }
  else if (t_type == 4) {
    if (lib_timing->rise_constraint == NULL && lib_timing->fall_constraint ==
NULL) {
```

```
      sprintf(msg,"constraint table is required for timing_type
%s.",type_name);
      libcerror(msg);
    }
  }
}

/* ================================================ */

public
void libc_time_piecewise(
      int piece,
      int field,
      float value)
{ libc_piece_value_rec *new,*p,*prev=NULL;

  /* ---- search first */
  for (p=lib_timing->piecewise;p!=NULL;prev=p,p=p->next) {
    if (p->piece == piece)
      goto insert_value;
    if (p->piece > piece)
      break;
  }
  /* ---- insert a new entry (fanout from small->large) */
  new = new_libc_piece_value_rec();
  new->piece                  = piece;
  /* ---- set default value */
  new->fall_delay_intercept   = tech_lib->default_fall_delay_intercept;
  new->rise_delay_intercept   = tech_lib->default_rise_delay_intercept;
  new->fall_nonpaired_twin    = tech_lib->default_fall_nonpaired_twin;
  new->rise_nonpaired_twin    = tech_lib->default_rise_nonpaired_twin;
  new->fall_pin_resistance    = tech_lib->default_fall_pin_resistance;
  new->rise_pin_resistance    = tech_lib->default_rise_pin_resistance;
  new->fall_wire_resistance   = tech_lib->default_fall_wire_resistance;
  new->rise_wire_resistance   = tech_lib->default_rise_wire_resistance;

  if (prev == NULL)
    lib_timing->piecewise = new;
  else
    prev->next = new;
  new->next = p;
  p = new;

insert_value :
  switch (field) {
    case 1 :      p->fall_delay_intercept = value;    break;
    case 2 :      p->fall_nonpaired_twin = value;           break;
    case 3 :      p->fall_pin_resistance = value;           break;
    case 4 :      p->fall_wire_resistance = value;    break;
    case 5 :      p->rise_delay_intercept = value;    break;
    case 6 :      p->rise_nonpaired_twin = value;           break;
    case 7 :      p->rise_pin_resistance = value;           break;
    case 8 :      p->rise_wire_resistance = value;    break;
    default :     assert(0);
  }
}
```

```
/* ================================================= */

public
float_buffer *libc_time_copy_float_buf(
      float_buffer *fbuf)
{ int size;
  float_buffer *result;

  if (fbuf == NULL)
    return(NULL);
  size = sizeof_float_buffer(fbuf);
  result = get_float_buffer(size);
  memcpy(result,fbuf,size*sizeof(float));
  return(result);
}

/* ================================================= */

/* ---- for both timing and power */

public
int libc_time_find_template(
      text_buffer *name,
      libc_lu_table_template_rec *head,
      libc_table_val_rec *tbl)
{ libc_lu_table_template_rec *p;
  int i,code = 0;       /* not found */

  if (strcmp(name,"scalar") == 0) {
    tbl->_tbl = NULL;          /* scalar table */
    code = 1;                  /* found */
  }
  else {
    for (p=head;p!=NULL;p=p->next) {
      if (strcmp(p->tt_name,name) == 0) {
        tbl->_tbl = p;
        code = 1;       /* found */
        /* ---- copy default index */
        if (p->index_1 != NULL) {
          tbl->index1 = get_float_buffer(sizeof_float_buffer(p->index_1));
          for (i=0;i<sizeof_float_buffer(p->index_1);i++)
            tbl->index1[i] = p->index_1[i];
        }
        if (p->index_2 != NULL) {
          tbl->index2 = get_float_buffer(sizeof_float_buffer(p->index_2));
          for (i=0;i<sizeof_float_buffer(p->index_2);i++)
            tbl->index2[i] = p->index_2[i];
        }
        if (p->index_3 != NULL) {
          tbl->index3 = get_float_buffer(sizeof_float_buffer(p->index_3));
          for (i=0;i<sizeof_float_buffer(p->index_3);i++)
            tbl->index3[i] = p->index_3[i];
        }
        break;
      }
    }
  }
```

```
  free_text_buffer(name);
  return(code);
}

/* ------------------------------------------------ */

public
void libc_time_values_handle(
      libc_table_val_rec *timing_tbl,
      libc_float_list_list *f_lists)
{ int size1,size2,size3;

  if (timing_tbl->_tbl == NULL) {         /* scalar table */
    if (f_lists->next != NULL || f_lists->v_list->next != NULL) {
      libcerror("only one value is allowed in scalar table.");
    }
    timing_tbl->scalar_val = f_lists->v_list->fvalue;
    free_libc_float_list_list(f_lists);
  }
  else {
    size1 = (timing_tbl->index1)? sizeof_float_buffer(timing_tbl->index1) :
0;
    size2 = (timing_tbl->index2)? sizeof_float_buffer(timing_tbl->index2) :
0;
    size3 = (timing_tbl->index3)? sizeof_float_buffer(timing_tbl->index3) :
0;
    timing_tbl->values =
libc_util_float_lists3buffer(size1,size2,size3,f_lists);
  }
}

/* ================================================ */

public
void libc_time_handle(
      int table_type)
{ libc_table_val_rec **tbl;

  switch (table_type) {
    case 0 :
      free_libc_table_val_rec(lib_timing_tbl);
      lib_timing_tbl = NULL;
      return;
    case 1 :      tbl = &(lib_timing->cell_rise);          break;
    case 2 :      tbl = &(lib_timing->cell_fall);          break;
    case 3 :      tbl = &(lib_timing->rise_propagation);   break;
    case 4 :      tbl = &(lib_timing->fall_propagation);   break;
    case 5 :      tbl = &(lib_timing->rise_transition);    break;
    case 6 :      tbl = &(lib_timing->fall_transition);    break;
    case 7 :      tbl = &(lib_timing->rise_constraint);    break;
    case 8 :      tbl = &(lib_timing->fall_constraint);    break;
  }
  free_libc_table_val_rec(*tbl);
  (*tbl) = lib_timing_tbl;
  lib_timing_tbl = NULL;
}
```

```
/* ------------------------------------------------ */

public
void libc_time_power_handle(
      int table_type)
{ libc_table_val_rec **tbl;

  switch (table_type) {
    case 0 :      tbl = &(lib_int_power->fall_power);       break;
    case 1 :      tbl = &(lib_int_power->rise_power);       break;
    case 2 :      tbl = &(lib_int_power->power);                  break;
  }
  free_libc_table_val_rec(*tbl);
  (*tbl) = lib_timing_tbl;
  lib_timing_tbl = NULL;
}

/* ================================================ */

public
void libc_time_minimum_period(
      int is_high,
      float value)
{ libc_min_pulse_width_rec *mpp,*nmpp;

  if (lib_pin->min_pulse_width == NULL) {
    nmpp = new_libc_min_pulse_width_rec();
    nmpp->when = NULL;
    lib_pin->min_pulse_width = nmpp;
  }
  else {
    for(mpp=lib_pin->min_pulse_width;mpp->next;mpp=mpp->next);
    if (mpp->when == NULL)
      nmpp = mpp;
    else {
      nmpp = new_libc_min_pulse_width_rec();
      mpp->next = nmpp;
    }
  }
  if (is_high)
    nmpp->constraint_high = value;
  else
    nmpp->constraint_low = value;
}

/* ================================================ */
```

```
/*====================================================


 =================================================== */

#include "libc_def.h"

/* ================================================= */

public
void libc_util_name_list1(
      YYSTYPE *r,
      YYSTYPE *r1)
{ libc_name_list_rec *np;

  np = new_libc_name_list_rec();
  np->name = r1->string;
  r->name_list.head = r->name_list._tail = np;
}

/* ------------------------------------------------- */

public
void libc_util_name_list2(
      YYSTYPE *r,
      YYSTYPE *r1,
      YYSTYPE *r3)
{ libc_name_list_rec *np;

  np = new_libc_name_list_rec();
  np->name = r3->string;
  r->name_list.head = r1->name_list.head;
  r1->name_list._tail->next = np;
  r->name_list._tail = np;
}

/* ================================================= */

public
void libc_util_float_list1(
      YYSTYPE *r,
      YYSTYPE *r1)
{ libc_float_list_rec *vp;

  vp = new_libc_float_list_rec();
  vp->fvalue = r1->real_val;
  r->float_list.head = r->float_list._tail = vp;
}

/* ------------------------------------------------- */

public
void libc_util_float_list2(
      YYSTYPE *r,
      YYSTYPE *r1,
      YYSTYPE *r3)
```

```
{ libc_float_list_rec *vp;

  vp = new_libc_float_list_rec();
  vp->fvalue = r3->real_val;
  r->float_list.head = r1->float_list.head;
  r1->float_list._tail->next = vp;
  r->float_list._tail = vp;
}

/* ================================================= */

public
void libc_util_float_list_list1(
      YYSTYPE *r,
      YYSTYPE *r1)
{ libc_float_list_list *vp;

  vp = new_libc_float_list_list();
  vp->v_list = r1->float_list.head;
  r->float_list_list.head = r->float_list_list._tail = vp;
}

/* ------------------------------------------------- */

public
void libc_util_float_list_list2(
      YYSTYPE *r,
      YYSTYPE *r1,
      YYSTYPE *r3)
{ libc_float_list_list *vp;

  vp = new_libc_float_list_list();
  vp->v_list = r3->float_list.head;
  r->float_list_list.head = r1->float_list_list.head;
  r1->float_list_list._tail->next = vp;
  r->float_list_list._tail = vp;
}

/* ================================================= */

public
any_unit_rec *libc_util_unit(
      float unit)
{ any_unit_rec *up;

  up = new_any_unit_rec();
  up->unit = unit;
  return(up);
}

/* ================================================= */

public
float_buffer *libc_util_float_list2buffer(
      libc_float_list_rec *f_list)
{ int i,size,is_inc=1,error=0;
  libc_float_list_rec *p;
```

```
  float_buffer *fbuf;
  float v1,v2;

  if (f_list == NULL)
    return(NULL);

  for(p=f_list,size=0;p!=NULL;p=p->next,size++);
  fbuf = get_float_buffer(size);
  if (size >= 2) {
    if (f_list->fvalue >= f_list->next->fvalue)
      is_inc = 1;
    else
      is_inc = 0;
  }

  for (p=f_list,i=0;p!=NULL;p=p->next,i++) {
    fbuf[i] = p->fvalue;
    if (i > 0) {
      if (is_inc) {
      if (fbuf[i-1] <= fbuf[i]) {
        v1 = fbuf[i-1];
        v2 = fbuf[i];
        error = 1;
      }
      }
      else {
      if (fbuf[i-1] >= fbuf[i]) {
        v1 = fbuf[i-1];
        v2 = fbuf[i];
        error = 1;
      }
      }
    }
  }
  free_libc_float_list_rec(f_list);
  if (error) {
    char msg[256];
    sprintf(msg,"Value %G shall %s than value %G in
index",v1,is_inc?"larger":"smaller",v2);
    libc_error(msg);
  }
  return(fbuf);
}

/* ------------------------------------------------ */

public
float_buffer **libc_util_float_lists2buffer(
      libc_float_list_list *f_lists)
{ int i,j,size1,size2;
  libc_float_list_list *q;
  libc_float_list_rec *p;
  float_buffer **fbuf;

  for (q=f_lists,size1=0;q!=NULL;q=q->next,size1++);
  fbuf = (float_buffer **) get_ptr_buffer(size1);
  for (p=f_lists->v_list,size2=0;p!=NULL;p=p->next,size2++);
```

```
  if (size1 == 1) {           /* 1D array */
    fbuf[0] = (float_buffer *) get_float_buffer(size2);
    for (p=f_lists->v_list,j=0;p!=NULL;p=p->next,j++)
      fbuf[0][j] = p->fvalue;
  }
  else {                /* 2D array */
    for (q=f_lists,i=0;q!=NULL;q=q->next,i++) {
      fbuf[i] = (float_buffer *) get_float_buffer(size2);
      for (p=q->v_list,j=0;p!=NULL;p=p->next,j++)
        fbuf[i][j] = p->fvalue;
    }
  }
  return(fbuf);
}

/* ------------------------------------------------ */

public
float_buffer *libc_util_float_lists3buffer(
      int size1,
      int size2,
      int size3,
      libc_float_list_list *f_lists)
{ int i,j,k;
  libc_float_list_list *q;
  libc_float_list_rec *p;
  float_buffer *fbuf;

  q = f_lists;
  p = q->v_list;

  if (size2 == 0) {           /* 1D array */
    fbuf = (float_buffer *) get_float_buffer(size1);
    for (i=0;q!=NULL;q=q->next) {
      for (p=q->v_list;p!=NULL;p=p->next,i++)
        fbuf[i] = p->fvalue;
    }
  }
  else if (size3 == 0) {      /* 2D array */
    fbuf = (float_buffer *) get_float_buffer(size1 * size2);
    for (i=0;i<size1;i++) {
      for (j=0;j<size2;j++) {
        fbuf[i*size2+j] = (p!= NULL)? p->fvalue : 0.0;
      p = p->next;
      if (p == NULL && q != NULL) {
        q = q->next;
        p = (q != NULL)? q->v_list : NULL;
      }
      }
    }
  }
  else {                /* 3D array */
    fbuf = (float_buffer *) get_float_buffer(size1 * size2 * size3);
    for (i=0;i<size1;i++) {
      for (j=0;j<size2;j++) {
        for (k=0;k<size3;k++) {
```

```
          fbuf[(i*size2+j)*size3+k] = (p!= NULL)? p->fvalue : 0.0;
        p = p->next;
        if (p == NULL && q != NULL) {
          q = q->next;
          p = (q != NULL)? q->v_list : NULL;
        }
      }
      }
    }
  }
  free_libc_float_list_list(f_lists);
  return(fbuf);
}

/* ================================================ */

public
libc_bool_opr_rec *libc_util_bool_opr(
      libc_bool_type_E op_type,
      int value)
{ libc_bool_opr_rec * bor;

  bor = new_libc_bool_opr_rec();
  bor->type = op_type;
  if (op_type == ID_B)
    bor->u.id_name = (text_buffer *) value;
  else
    bor->u.value = value;
  return(bor);
}

/* ================================================ */

public
void libc_util_wire_load_fanout(
      int fanout,
      float length,
      float ave_cap,
      float std_dev,
      int no_of_net)
{ libc_fanout_length_rec *new,*p,*prev;

  new = new_libc_fanout_length_rec();
  new->fanout      = fanout;
  new->length      = length;
  new->area        = lib_wire_load->area;
  new->capacitance = lib_wire_load->capacitance;
  new->resistance  = lib_wire_load->resistance;
  new->ave_cap     = ave_cap;
  new->std_dev     = std_dev;
  new->no_of_net = no_of_net;

  /* ---- sort by fanout (small->large) */
  for (p=lib_wire_load->fanout_length,prev=NULL;p!=NULL;prev=p,p=p->next) {
    if (p->fanout >= new->fanout)
      break;
  }
```

```
  new->next = p;
  if (prev == NULL)
    lib_wire_load->fanout_length = new;
  else
    prev->next = new;
}

/* ---------------------------------------------- */

public
void libc_util_wire_load_table(
      int fanout,
      int field,  /* 1: length,2: capacitance, 3: resistance, 4: area */
      float value)
{ libc_fanout_length_rec *new,*p,*prev=NULL;

  /* ---- search first */
  for (p=lib_wire_load->fanout_length;p!=NULL;prev=p,p=p->next) {
    if (p->fanout == fanout)
      goto insert_value;
    if (p->fanout > fanout)
      break;
  }
  /* ---- insert a new entry (fanout from small->large) */
  new = new_libc_fanout_length_rec();
  new->fanout           = fanout;
  new->area       = lib_wire_load->area;              /* default value */
  new->capacitance      = lib_wire_load->capacitance;       /* default value
*/
  new->resistance = lib_wire_load->resistance;        /* default value */
  if (prev == NULL)
    lib_wire_load->fanout_length = new;
  else
    prev->next = new;
  new->next = p;
  p = new;

insert_value :
  switch (field) {
    case 1 :      p->length = value;            break;
    case 2 :      p->capacitance = value;       break;
    case 3 :      p->resistance = value;        break;
    case 4 :      p->area = value;        break;
    default :     assert(0);
  }
}

/* ================================================ */

public
void libc_util_wl_select(
      float min_area,
      float max_area,
      text_buffer *name)
{ libc_wire_load_rec *p;
  libc_wire_load_from_area_rec *wap,*prev,*new;
  char msg[100];
```

```
  for (p=tech_lib->wire_load;p!=NULL;p=p->next) {
    if (strcmp(p->wl_name,name) == 0) {
      free_text_buffer(name);
      new = new_libc_wire_load_from_area_rec();
      new->min_area = min_area;
      new->max_area = max_area;
      new->_load_model = p;

      /* ---- sort by area (and each group shall not overlap) */
      for (wap=lib_wire_load_sel->area_table,prev=NULL;
         wap!=NULL;
         prev=wap,wap=wap->next) {
        if (wap->min_area >= min_area)
        break;
      }
      new->next = wap;
      if (prev == NULL)
      lib_wire_load_sel->area_table = new;
      else
        prev->next = new;
      return;
    }
  }
  sprintf(msg,"wire load model(%s) not found.",name);
  libcerror(msg);
  free_text_buffer(name);
}

/* ------------------------------------------------ */

/* ================================================ */

/* ------------------------------------------------ */

/* ================================================ */
```

```
#ifndef YYSTYPE
#define YYSTYPE int
#endif
#define     K_LIBRARY   258
#define     K_ANPPP     259
#define     K_BNS 260
#define     K_COMMENT   261
#define     K_CU  262
#define     K_DATE      263
#define     K_DM  264
#define     K_IPSM      265
#define     K_LPU 266
#define     K_MWE 267
#define     K_MDL 268
#define     K_NP  269
#define     K_NT  270
#define     K_NV  271
#define     K_NTIDF     272
#define     K_NPPP      273
#define     K_PT  274
#define     K_PU  275
#define     K_POPSRC    276
#define     K_POPV      277
#define     K_PIPV      278
#define     K_PRU 279
#define     K_RC  280
#define     K_REVISION  281
#define     K_SIMULATION      282
#define     K_TU  283
#define     K_UPP 284
#define     K_VU  285
#define     K_WLF 286
#define     K_DCLP      287
#define     K_DCP 288
#define     K_DCC 289
#define     K_DF0 290
#define     K_DF1 291
#define     K_DR0 292
#define     K_DR1 293
#define     K_DEC 294
#define     K_DFDI      295
#define     K_DFNT      296
#define     K_DFPR      297
#define     K_DFWR      298
#define     K_DFWE      299
#define     K_DFWI      300
#define     K_DFL 301
#define     K_DIOPC     302
#define     K_DIOFR     303
#define     K_DIORR     304
#define     K_DIPC      305
#define     K_DIF 306
#define     K_DIR 307
#define     K_DMC 308
#define     K_DMFO      309
#define     K_DMT 310
#define     K_DMU 311
```

```
#define     K_DMP 312
#define     K_DOC 313
#define     K_DOPC      314
#define     K_DOFR      315
#define     K_DORR      316
#define     K_DPL 317
#define     K_DPP 318
#define     K_DRFC      319
#define     K_DRRC      320
#define     K_DRDI      321
#define     K_DRNT      322
#define     K_DRPR      323
#define     K_DRWR      324
#define     K_DRWE      325
#define     K_DRWI      326
#define     K_DSC 327
#define     K_DSF 328
#define     K_DSR 329
#define     K_DWL 330
#define     K_DWLA      331
#define     K_DWLC      332
#define     K_DWLM      333
#define     K_DWLR      334
#define     K_DWLS      335
#define     P_CR  336
#define     P_CF  337
#define     P_CLP 338
#define     P_CP  339
#define     P_DC  340
#define     P_DF  341
#define     P_DR  342
#define     P_FDI 343
#define     P_FPR 344
#define     P_FP  345
#define     P_FT  346
#define     P_FWR 347
#define     P_FWE 348
#define     P_FWI 349
#define     P_HF  350
#define     P_HR  351
#define     P_IP  352
#define     P_IF  353
#define     P_IR  354
#define     P_MP  355
#define     P_MPWH      356
#define     P_MPWL      357
#define     P_NF  358
#define     P_NR  359
#define     P_PC  360
#define     P_PP  361
#define     P_RF  362
#define     P_RR  363
#define     P_REF 364
#define     P_RER 365
#define     P_RDI 366
#define     P_RPR 367
#define     P_RP  368
```

```
#define     P_RT   369
#define     P_RWR 370
#define     P_RWE 371
#define     P_RWI 372
#define     P_SF   373
#define     P_SR   374
#define     P_SKF 375
#define     P_SKR 376
#define     P_SLF 377
#define     P_SLR 378
#define     P_WC   379
#define     P_WR   380
#define     K_CLU 381
#define     K_DEFINE      382
#define     K_DCA 383
#define     K_LF   384
#define     K_PD   385
#define     K_RL   386
#define     K_TECH        387
#define     K_CELL        388
#define     K_IV   389
#define     K_LTT 390
#define     K_OC   391
#define     K_OV   392
#define     K_PCELL       393
#define     K_PLT 394
#define     K_PS   395
#define     K_RTD 396
#define     K_FTD 397
#define     K_SC   398
#define     K_SF   399
#define     K_TR   400
#define     K_TYPE        401
#define     K_WL   402
#define     K_WLS 403
#define     K_WLT 404
#define     AREA   405
#define     C_APC 406
#define     C_CF   407
#define     C_POWER       408
#define     C_LP   409
#define     C_CC   410
#define     C_DF   411
#define     C_DT   412
#define     C_GP   413
#define     C_DU   414
#define     C_HNC 415
#define     C_IT   416
#define     C_MO   417
#define     C_PC   418
#define     C_PT   419
#define     C_PL   420
#define     C_P    421
#define     C_SF   422
#define     C_SG   423
#define     C_SBD 424
#define     C_VN   425
```

```
#define     C_PE  426
#define     C_PO  427
#define     C_RC  428
#define     C_BUNDLE    429
#define     C_BUS 430
#define     C_FF  431
#define     C_FFB 432
#define     C_IP  433
#define     C_LPG 434
#define     C_LATCH     435
#define     C_LB  436
#define     C_LUT 437
#define     C_MEM 438
#define     C_PIN 439
#define     C_RT  440
#define     C_STATE     441
#define     C_ST  442
#define     C_TC  443
#define     MEMBERS     444
#define     BUS_TYPE    445
#define     MEM_READ    446
#define     MEM_WRITE   447
#define     CE_PROPERTY 448
#define     CE_DE 449
#define     CE_PP 450
#define     PP_PROPERTIES     451
#define     PP_DISABLE  452
#define     CLOCK_ON    453
#define     NEXT_ST     454
#define     CLEAR 455
#define     PRESET      456
#define     CL_PS_V1    457
#define     CL_PS_V2    458
#define     ON_ALSO     459
#define     ENABLE      460
#define     DATA_IN     461
#define     FORCE_01    462
#define     FORCE_10    463
#define     FORCE_00    464
#define     FORCE_11    465
#define     REL_INP     466
#define     REL_INPS    467
#define     REL_OUTP    468
#define     VALUES      469
#define     IP_EOOO     470
#define     IP_PL 471
#define     IP_FP 472
#define     IP_RP 473
#define     IP_POWER    474
#define     VALUE 475
#define     LUT_IP      476
#define     CM_TYPE     477
#define     CM_RAM      478
#define     CM_ROM      479
#define     CM_ADDR_WIDTH     480
#define     CM_WORD_WIDTH     481
#define     CM_C_ADDR   482
```

```
#define     CM_R_ADDR    483
#define     PIN_CAP      484
#define     PIN_CLK      485
#define     PIN_CGEP     486
#define     PIN_CC       487
#define     PIN_DIR      488
#define     PIN_DF       489
#define     PIN_DC       490
#define     PIN_DT       491
#define     PIN_ERBF0    492
#define     PIN_ERBF1    493
#define     PIN_ERBR0    494
#define     PIN_ERBR1    495
#define     PIN_ERF      496
#define     PIN_ERR      497
#define     PIN_ERLF     498
#define     PIN_ERLR     499
#define     PIN_EC       500
#define     PIN_FCSAT    501
#define     PIN_FCSBT    502
#define     PIN_FTAT     503
#define     PIN_FTBT     504
#define     PIN_FWE      505
#define     PIN_FWI      506
#define     PIN_FL       507
#define     PIN_FUNCTION       508
#define     PIN_H 509
#define     PIN_IM       510
#define     PIN_ISL      511
#define     PIN_IV       512
#define     PIN_IN       513
#define     PIN_IO       514
#define     PIN_IP       515
#define     PIN_MAX_FO   516
#define     PIN_MAX_TRANS      517
#define     PIN_MAX_CAP 518
#define     PIN_MIN_FO   519
#define     PIN_MIN_TRANS      520
#define     PIN_MIN_CAP 521
#define     PIN_MP       522
#define     PIN_MPWH     523
#define     PIN_MPWL     524
#define     PIN_MPP      525
#define     PIN_MDL      526
#define     PIN_NST      527
#define     PIN_OSL      528
#define     PIN_OV       529
#define     PIN_PFT      530
#define     PIN_PP       531
#define     PIN_PT       532
#define     PIN_PO       533
#define     PIN_PC       534
#define     PIN_PR       535
#define     PIN_RC       536
#define     PIN_RCSAT    537
#define     PIN_RCSBT    538
#define     PIN_RTAT     539
```

```
#define     PIN_RTBT    540
#define     PIN_RWE     541
#define     PIN_RWI     542
#define     PIN_SC      543
#define     PIN_SF      544
#define     PIN_TS      545
#define     PIN_VN      546
#define     PIN_WC      547
#define     PIN_WCC     548
#define     PIN_XF      549
#define     TIMING      550
#define     MIN_PLUSE_WIDTH   551
#define     MIN_PERIOD  552
#define     PIN_IPO     553
#define     ADDRESS     554
#define     TRACKS      555
#define     TRACK_AREA  556
#define     TABLE 557
#define     CTC_PIN     558
#define     CTC_DIR     559
#define     CTC_FUNC    560
#define     CTC_ST      561
#define     CTC_TOO     562
#define     TI_ERSF0    563
#define     TI_ERSF1    564
#define     TI_ERSR0    565
#define     TI_ERSR1    566
#define     TI_FR 567
#define     TI_RR 568
#define     TI_IF 569
#define     TI_IR 570
#define     TI_RBP      571
#define     TI_ROP      572
#define     TI_RP 573
#define     TI_SDF_C    574
#define     TI_SDF_CS   575
#define     TI_SDF_CE   576
#define     TI_SDF_E    577
#define     TI_SF 578
#define     TI_SR 579
#define     TI_TT 580
#define     TI_TS 581
#define     TI_WHEN     582
#define     TI_WS 583
#define     TI_WE 584
#define     TI_FDI      585
#define     TI_FNT      586
#define     TI_FPR      587
#define     TI_FWR      588
#define     TI_RDI      589
#define     TI_RNT      590
#define     TI_RPR      591
#define     TI_RWR      592
#define     CELL_DEGR   593
#define     CELL_FALL   594
#define     CELL_RISE   595
#define     R_PROP      596
```

```
#define     F_PROP      597
#define     R_TRANS     598
#define     F_TRANS     599
#define     R_CONS      600
#define     F_CONS      601
#define     NO_EDGE     602
#define     BOTH_EDGES  603
#define     START_EDGE  604
#define     END_EDGE    605
#define     MPW_CH      606
#define     MPW_CL      607
#define     MP_C  608
#define     IV_L  609
#define     IV_H  610
#define     IV_MIN      611
#define     IV_MAX      612
#define     TBL_VAR1    613
#define     TBL_VAR2    614
#define     TBL_VAR3    615
#define     TBL_IDX1    616
#define     TBL_IDX2    617
#define     TBL_IDX3    618
#define     LTT_INT     619
#define     LTT_TONC    620
#define     LTT_ONL     621
#define     LTT_ONWC    622
#define     LTT_ONPC    623
#define     LTT_ROTONC  624
#define     LTT_ROONL   625
#define     LTT_ROONWC  626
#define     LTT_ROONPC  627
#define     LTT_CPT     628
#define     LTT_RPT     629
#define     LTT_OPT     630
#define     LTT_CD      631
#define     OC_PROCESS  632
#define     OC_TEMP     633
#define     OC_TREE     634
#define     OC_VOLT     635
#define     OC_PR 636
#define     OV_L  637
#define     OV_H  638
#define     OV_MIN      639
#define     OV_MAX      640
#define     PS_DPR      641
#define     PC_CELL_ENUM      642
#define     TR_FF 643
#define     TR_SF 644
#define     TYPE_BASE   645
#define     TYPE_ARRAY  646
#define     TYPE_FROM   647
#define     TYPE_TO     648
#define     TYPE_WIDTH  649
#define     TYPE_DT     650
#define     TYPE_BIT    651
#define     TYPE_DOWNTO 652
#define     WL_RES      653
```

```
#define     WL_SLOPE    654
#define     WL_FL 655
#define     WLS_WLFA    656
#define     WLT_AREA    657
#define     WLT_CAP     658
#define     WLT_RES     659
#define     BOOL_T      660
#define     BOOL_F      661
#define     CU_A  662
#define     CU_MA 663
#define     CU_UA 664
#define     LU_MF 665
#define     LU_UF 666
#define     LU_NF 667
#define     LU_PF 668
#define     LU_FF 669
#define     VU_V  670
#define     VU_MV 671
#define     PU_W  672
#define     PU_MW 673
#define     PU_UW 674
#define     PU_NW 675
#define     PU_PW 676
#define     RU_OHM      677
#define     RU_KOHM     678
#define     TU_NS 679
#define     TU_PS 680
#define     DM_G_ECL    681
#define     DM_G_CMOS   682
#define     DM_TBL_LOOKUP     683
#define     DM_CMOS2    684
#define     DM_P_COMS   685
#define     IPO_MF      686
#define     IPO_NS      687
#define     IPO_IF      688
#define     MMP_MAX     689
#define     MMP_MIN     690
#define     MMP_PLUS    691
#define     PT_LENGTH   692
#define     PT_WIRE_CAP 693
#define     PT_PIN_CAP  694
#define     PT_TOTAL_CAP      695
#define     WLF_WAND    696
#define     WLF_WOR     697
#define     WLM_T 698
#define     WLM_S 699
#define     WLM_E 700
#define     RT_PS 701
#define     RT_PDS      702
#define     RT_PIDS     703
#define     RT_PODS     704
#define     TT_BEST     705
#define     TT_BAL      706
#define     TT_WORST    707
#define     SA_SA0      708
#define     SA_SA1      709
#define     SA_SA01     710
```

```
#define     DIR_INPUT   711
#define     DIR_OUTPUT  712
#define     DIR_INOUT   713
#define     DIR_INTERNAL      714
#define     DRT_PULL_UP 715
#define     DRT_PULL_DOWN     716
#define     DRT_OPEN_DRAIN    717
#define     DRT_OPEN_SOURCE   718
#define     DRT_BUS_HOLD      719
#define     DRT_RES     720
#define     DRT_RES0    721
#define     DRT_RES1    722
#define     N_DATA      723
#define     N_PRESET    724
#define     N_CLEAR     725
#define     N_LOAD      726
#define     N_SCAN_IN   727
#define     N_SCAN_ENABLE     728
#define     CLK_ENABLE  729
#define     ACT_HIGH    730
#define     ACT_LOW     731
#define     ACT_RISING  732
#define     ACT_FALLING 733
#define     NONE_SC     734
#define     LOW_SC      735
#define     MED_SC      736
#define     HIGH_SC     737
#define     TIT_RE      738
#define     TIT_FE      739
#define     TIT_PS      740
#define     TIT_CL      741
#define     TIT_HR      742
#define     TIT_HF      743
#define     TIT_SR      744
#define     TIT_SF      745
#define     TIT_RR      746
#define     TIT_RF      747
#define     TIT_TSD     748
#define     TIT_TSE     749
#define     TIT_RMR     750
#define     TIT_RMF     751
#define     TIT_C 752
#define     TIT_SKR     753
#define     TIT_SKF     754
#define     TIT_NSHR    755
#define     TIT_NSHF    756
#define     TIT_NSSR    757
#define     TIT_NSSF    758
#define     TIT_NCHH    759
#define     TIT_NCHL    760
#define     TIT_NCLH    761
#define     TIT_NCLL    762
#define     TIS_POS     763
#define     TIS_NEG     764
#define     TIS_NON     765
#define     ST_TSI      766
#define     ST_TSII     767
```